JÜRGEN BAYER

Das C# 2008 Codebook



Auf der Interaktiv-CD

- Durchsuchbares HTML-Repository
- Umfangreiche Artikel zu API, WMI, ADSI, WPF, LINQ und LINQ to SQL
- Weblog mit Aktualisierungen
- eBook

Auf der DVD

- Visual Studio Professional 2008 Edition (90-Tage-Testversion)

ADDISON-WESLEY

interaktiv CD DVD

Das C# 2008 Codebook

Unser Online-Tipp
für noch mehr Wissen …

informit.de

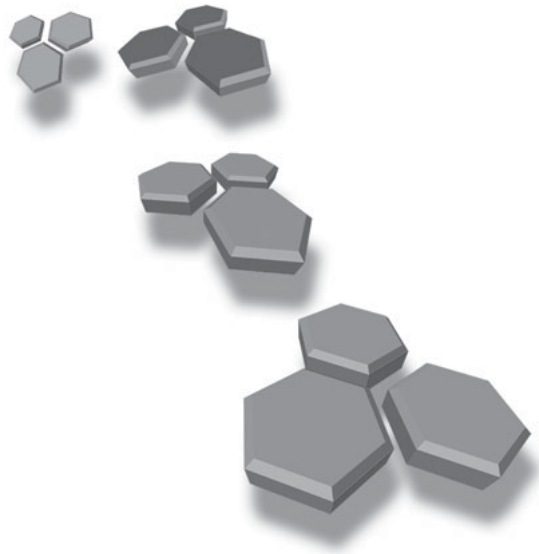
Aktuelles Fachwissen rund um die Uhr
– zum Probelesen, Downloaden oder
auch auf Papier.

www.informit.de



Jürgen Bayer

Das C# 2008 Codebook



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt. Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ®-Symbol in diesem Buch nicht verwendet.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt. Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1
10 09 08

ISBN 973-3-8273-2576-1

© 2008 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten

Korrektorat: Simone Meißner
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Herstellung: Elisabeth Prümm, epruemm@pearson.de
Satz: Kösel, Krugzell (www.KoeselBuch.de)
Umschlaggestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Druck und Verarbeitung: Kösel Krugzell (www.KoeselBuch.de)

Printed in Germany

Inhaltsverzeichnis

Vorwort 17

Teil I Einführung 19

Zum Buch 21

Der Inhalt 21
Einführende Artikel 21
Die Rezepte 21
Die Buch-CD 23
Das Codebook-Klassenbibliotheks-Projekt 23
Das Repository, die Beispiele und Komponenten 24
Unit-Testing 24
Das Buch-Blog 24
Der Index 24
Voraussetzungen 25
Typografische Konventionen und Icons 25

Recherche-Möglichkeiten 27

Wichtige .NET-Websites 27
Newsgroup-Recherche bei Google 30
Suche bei Microsoft 33
Debugging des .NET-Framework-Quellcodes 35
Der .NET Reflector von Lutz Roeder 39

Teil II Rezepte 41

Basics 43

1 Zahlen formatieren 43
2 Negative und positive Zahlen unterschiedlich formatieren 45
3 (Ganz-)Zahlen hexadezimal darstellen 45
4 Hexadezimale Zahlen in dezimale Zahlen konvertieren 46
5 Ganzzahlen dual darstellen 46
6 Dualzahlen in Ganzzahlen konvertieren 51
7 Zahlen in verschiedenen Notationen darstellen 52
8 Zahlen kaufmännisch runden 54
9 Den kleinsten und den größten Wert eines numerischen Datentyps ermitteln 55
10 Gradmaß in das Bogenmaß umrechnen 55
11 Bogenmaß in das Gradmaß umrechnen 55
12 Umstellen der aktuellen Kultur 56
13 Die verfügbaren Kulturen auslesen 57
14 Zahlen in verschiedenen Länderformaten darstellen 60

15 Einen Bytewert in ein besser lesbares Format umwandeln 60
16 Überprüfen, ob ein String eine gültige Zahl speichert 61
17 Überprüfen, ob ein String ein gültiges Datum speichert 65
18 Zufallszahlen erzeugen 66
19 High- und Low-Word lesen 69
20 High- und Low-Word setzen 70
21 Beep ausgeben 71
22 Strings ohne Berücksichtigung der Groß- bzw. Kleinschreibung vergleichen 72
23 Teilstrings flexibel ersetzen 72
24 Strings an vorgegebenen Trennzeichen auftrennen 79
25 Mehrere (Leer-)Zeichen in ein (Leer-)Zeichen konvertieren 80
26 Erstes Zeichen aller Wörter im String groß- und die restlichen Zeichen kleinschreiben 81
27 Anzahl der Wörter in einem String ermitteln 82
28 Alle Wörter eines Strings extrahieren 85
29 Einen String an einem rechten Rand wortgerecht umbrechen 86
30 String auf die ersten Wörter bis zu einer Maximallänge kürzen 89
31 Teilstring nur dann ersetzen, wenn dieser am Anfang eines Strings gespeichert ist 91
32 Linken und rechten Teilstring extrahieren 93
33 Zahlen aus einem String extrahieren 94
34 Überprüfen, ob ein String mit einer Zahl beginnt 95
35 Zufalls-String berechnen 97
36 Konstantennamen einer Aufzählung als String auslesen 99
37 Strings in Aufzählungswerte konvertieren 100
38 String in Byte-Array umwandeln 101
39 Byte-Array in String lesen 104
40 Auflistungen und Arrays sortieren 105
41 Auflistungen und Arrays performant durchsuchen 109
42 Typisierte Auflistungen erzeugen 115
43 Schnelle Auflistung mit Schlüssel- und Indexzugriff 120
44 Das Programm für eine definierte Zeit anhalten 121
45 Die Nachrichten einer Exception und ihrer inneren Exceptions ermitteln 122
46 Exceptions in Anwendungen korrekt auswerten 126

Datum und Zeit 131

47 Aktuelles Datum und aktuelle Uhrzeit ermitteln 131
48 Datumswerte formatieren 131
49 Datumswerte in verschiedenen Länderformaten ausgeben 133
50 Datum in das ISO-Format konvertieren 136
51 Datumswerte vergleichen 137
52 Die Differenz zwischen zwei Datumswerten berechnen 138
53 Alter in ganzen Jahren berechnen 145
54 Datumswerte addieren und subtrahieren 145
55 Eine Zeitangabe in einem String in ein Datum mit der Basis DateTime.Min umwandeln 147

56 Die Anzahl der Tage eines Monats berechnen 148
57 Die Anzahl der Tage eines Jahres berechnen 148
58 Den Wochentag eines Datums ermitteln 149
59 Den Namen eines Wochentags ermitteln 150
60 Den Namen eines Monats ermitteln 152
61 Den ersten und letzten Tag der Woche eines Datums berechnen 153
62 Den ersten und den letzten Tag eines Monats berechnen 154
63 Das Quartal eines gegebenen Datums berechnen 157
64 Den ersten und den letzten Tag eines Quartals berechnen 157
65 Ermitteln, ob ein Jahr ein Schaltjahr ist 160
66 Ermitteln der ISO-Kalenderwoche 161
67 Ermitteln, mit welchem Datum eine Kalenderwoche beginnt 165
68 Anzahl der Kalenderwochen eines Jahres bestimmen 168
69 Systemdatum und -zeit setzen 169
70 Atomuhr-Datum inklusive Uhrzeit von einem NIST-Server abfragen 172
71 Datum normalisieren 177
72 Deutsche Feiertage und andere besondere Tage berechnen 181
73 Ermitteln, ob ein bestimmter Tag ein Feiertag ist 192
74 Zeit genau messen 194
75 Eine Eingabe daraufhin überprüfen, ob diese ein Datum ergeben kann 197
76 Eine Eingabe daraufhin überprüfen, ob diese eine Zeit ergeben kann 201
77 Datumswerte fehlerfrei zwischen Systemen mit verschiedenen Zeitzonen austauschen 202
78 Mit Zeitzonen arbeiten 204
79 Mit Kalendersystemen arbeiten 207

Anwendungen, Anwendungs-Konfiguration, Prozesse und Dienste 215

80 Den Dateinamen der Anwendung ermitteln 215
81 Das Verzeichnis der Anwendung ermitteln 216
82 Befehlszeilenargumente auswerten 216
83 User Account Control (UAC) berücksichtigen 219
84 Ausnahmen global behandeln 223
85 Den Speicherverbrauch von .NET-Anwendungen verstehen und gegebenenfalls reduzieren 226
86 Konfigurationsdaten in der .config-Datei verwalten 230
87 Konfigurationsdaten aus dem appSettings-Element der .config-Datei lesen 234
88 Externe Konfigurationsdateien lesen und schreiben 235
89 Konfigurationsdaten in eigenen Abschnitten speichern 240
90 Anwendungskonfiguration über eine eigene XML-Datei 245
91 Daten aus der Registry lesen und in die Registry schreiben 252
92 Verhindern, dass eine Anwendung mehrfach gestartet werden kann 260
93 Aktivieren einer laufenden Anwendung 262
94 Auflisten aller laufenden Prozesse eines Computers 269
95 Auflisten aller laufenden Anwendungen eines Computers 271
96 Informationen über die Dienste eines Computers auslesen 272

97 Dienste starten, anhalten, fortfahren und stoppen 276
98 Ermitteln, ob eine Anwendung ausgeführt wird 282
99 Anwendungen starten 283
100 Dateien mit der assoziierten Anwendung öffnen 284
101 Starten einer anderen Anwendung und warten, bis diese beendet ist 285
102 Konsolenanwendungen starten und die Ausgabe auswerten 287
103 Andere Anwendungen beenden 289
104 Java-Anwendungen starten 291
105 Die Betätigung der Tastatur simulieren 296

Dateisystem 299

106 Dateiname einer Windowsanwendung ermitteln 299
107 Dateiname einer Klassenbibliothek ermitteln 300
108 Ordner einer Windowsanwendung ermitteln 301
109 Ordner einer Klassenbibliothek ermitteln 302
110 Dateiname für eine temporäre Datei erzeugen 302
111 Ordner für temporäre Dateien ermitteln 303
112 Windows-(Spezial-)Ordner auslesen 303
113 Das .NET-Framework-Verzeichnis ermitteln 306
114 Dateiname mit anderer Endung ermitteln 307
115 Relativen Pfad aus einem absoluten Pfad ermitteln 307
116 Absoluten Pfad aus einem relativen Pfad ermitteln 309
117 Überprüfen, ob eine Pfadangabe gültig ist 310
118 Die logischen Laufwerke des Systems ermitteln 311
119 Den Typ der Laufwerke eines Systems ermitteln 313
120 Alle Laufwerke eines bestimmten Typs ermitteln 314
121 Größe und freien Speicherplatz eines Laufwerks ermitteln 316
122 Überprüfen, ob ein Ordner existiert 317
123 Ordner erzeugen 318
124 Ordner mit .NET-Features kopieren 318
125 Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage über SHFileOperation kopieren 321
126 Ordner mit .NET-Features ohne Abbruch bei einer Ausnahme kopieren 328
127 Ordner umbenennen 336
128 Ordner verschieben 337
129 Ordner löschen 337
130 Ordnergröße ermitteln 338
131 Ordnergröße aller direkten Unterordner ermitteln 340
132 Überprüfen, ob eine Datei existiert 344
133 Dateien in einem Ordner und seinen Unterordnern suchen 344
134 Programmdateien in den Systempfaden suchen 346
135 Dateien kopieren 349
136 Dateien umbenennen 351
137 Dateien verschieben 351
138 Dateien löschen 352
139 Dateien vergleichen 353
140 Ordner vergleichen 355

141 Die Größe einer Datei ermitteln 358
142 Dateiattribute auslesen 359
143 Dateiattribute (z. B. den Schreibschutz) setzen oder aufheben 361
144 Die Erstell- und Zugriffsdaten einer Datei lesen und setzen 362
145 Die Version einer Datei auslesen 363
146 Den kurzen (8.3-)Dateinamen einer Datei auslesen 364
147 Alle Dateien eines Ordners auflisten 366
148 Dateien und Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage umbenennen oder verschieben 367
149 Dateien und Ordner in den Papierkorb verschieben 371
150 Größe des Papierkorbs und Anzahl der Dateien im Papierkorb ermitteln 374
151 Verknüpfungen anlegen 377
152 Das Dateisystem überwachen 380

Text-, binäre und Zip-Dateien 383

153 Textdateien lesen 383
154 Textdateien schreiben 385
155 Texte an Textdateien anfügen 387
156 Textdateien ändern 389
157 Binäre Dateien lesen 390
158 Binäre Dateien schreiben 392
159 Binäre Dateien Base64-codieren 393
160 Base64-codierte Strings in Dateien umwandeln 394
161 Komprimieren von Daten mit .NET-Klassen 395
162 Dekomprimieren von Daten mit .NET-Klassen 399
163 (ZIP-)Komprimieren von Daten mit #ziplib 401
164 (ZIP-)Dekomprimieren von Daten mit #ziplib 407
165 Dateien in ZIP-Archive komprimieren 409
166 (ZIP-)Archive aus einem Ordner erzeugen 412
167 Dateiinformationen aus (ZIP-)Archiven auslesen 415
168 (ZIP-)Archive entpacken 417

XML 423

169 XML-Dokumente über LINQ to XML lesen 423
170 XML-Dokumente mit Namensraum über LINQ to XML lesen 426
171 XML-Dokumente über ein XmlReader-Objekt lesen 429
172 XML-Dokumente über das X-DOM lesen 433
173 XML-Dokumente beim Einlesen gegen ein Schema (oder DTD) prüfen 436
174 XML-Dokumente auf Gültigkeit überprüfen 441
175 XML-Dokumente validierend über das X-DOM lesen 451
176 Gezielt über XPath auf Elemente in einem XML-Dokument ohne Namensraum zugreifen 453
177 Gezielt über XPath auf Elemente in einem XML-Dokument mit Namensraum zugreifen 455
178 XML-Dokumente mit einem XmlWriter erzeugen 457
179 XML-Dokumente über das X-DOM erzeugen und ändern 460
180 XML-Dokumente über LINQ to XML erzeugen 463
181 XML-Dokumente über ein DataTable- oder DataSet-Objekt erzeugen 465

182 XML-Dokumente über ein DataTable- oder ein DataSet-Objekt ändern 467
183 Binäre Daten in einem XML-Dokument speichern 469
184 Base64-codierte Bilder (binäre Daten) aus XML-Dokumenten auslesen 470
185 XML-Dokumente über LINQ to XML transformieren 471
186 XML-Dokumente über XSL-Dokumente transformieren 473

System 477

187 Den Namen des Computers ermitteln 477
188 Systempfade herausfinden 477
189 Windows-Version auslesen 480
190 Windows-Hauptversion ermitteln 480
191 Die Prozessorgeschwindigkeit ermitteln 482
192 Informationen über den Arbeitsspeicher auslesen 483
193 Die Version des aktuellen Service-Packs ermitteln 486
194 Installierte Programme auflisten 488
195 Informationen zu den parallelen Schnittstellen des Systems auslesen 491
196 Informationen zu den seriellen Schnittstellen des Systems auslesen 496
197 Ermitteln der Soundkarten des Systems 502
198 Windows herunterfahren und neu starten 503
199 Die verfügbaren Leistungsindikatoren auslesen 505
200 Den Arbeitsspeicher überwachen 511
201 Speicherauslastung und CPU-Belastung des aktuellen Prozesses überwachen 514
202 Speicherauslastung und CPU-Belastung des aktuellen Prozesses in ein Trace-Protokoll schreiben 516
203 Eigene Leistungsindikatoren implementieren 522
204 Monitor abschalten, in den Energiesparmodus schalten und wieder einschalten 527
205 System-Hotkeys registrieren und auswerten 529

Internet 541

206 Status der Internetverbindung abfragen 541
207 Pingen – Ermitteln, ob eine Internetverbindung besteht 544
208 TCP- und ICMP-Prüfsumme berechnen 546
209 Internetverbindung öffnen und schließen 548
210 Die IP-Adressen des Computers herausfinden 552
211 E-Mails über einen SMTP-Server versenden 554
212 E-Mails über MAPI bzw. Outlook versenden 560
213 Browser starten 562
214 Webseiten (HTML-Dokumente) in der Anwendung darstellen 563
215 Dateien von einem Web- oder FTP-Server über eine WebClient-Instanz downloaden 569
216 Dateien von einem Webserver über eine HttpWebRequest-Instanz downloaden 578
217 Intelligenter Up- und Download mit BITS 601
218 Die Größe einer Datei von einem FTP-Server abfragen 612
219 Dateien zu einem Web- oder FTP-Server hochladen 614
220 FTP-Verzeichnisse auslesen 621

221 Dateien und Ordner auf einem FTP-Server löschen 631
222 Ordner auf einem FTP-Server erstellen 632

Windows.Forms 635

223 Formulare ohne Titelleiste 635
224 (Rahmenlose) Formulare über den Clientbereich verschiebbar machen 635
225 Unbewegbare Formulare mit Titelleiste 637
226 Andockende Formulare 638
227 Ändern des Systemmenüs – Entfernen des Schließen-Eintrags und -Schalters 639
228 Formulare verlaufend füllen 640
229 Formulare mit speziellen Formen 641
230 Formulare mit dem Vista-Glas-Effekt ausstatten 647
231 Splash-Formulare 652
232 Ermitteln, auf welchen Monitoren ein Formular angezeigt wird 654
233 Ein Formular auf einem sekundären Bildschirm öffnen 658
234 Die Bildschirm-Position eines Steuerelements ermitteln 659
235 Die optimale Position eines Formulars oder eines Steuerelements bezogen auf ein Steuerelement ermitteln 660
236 Das Hauptformular einer Anwendung ermitteln 664
237 Menüs in der Laufzeit erstellen oder erweitern 665
238 Steuerelemente mit transparentem Hintergrund 667
239 TextBox automatisch beim Eintritt selektieren 668
240 TextBox auf Zahleingaben beschränken 670
241 Auf Return in einer TextBox reagieren 675
242 Bei der Betätigung der Return-Taste die Tab-Taste simulieren 676
243 Die Position des Eingabecursors in einer TextBox, RichTextBox oder MaskedTextBox ermitteln 678
244 Die angezeigten Zeilen einer MultiLine-TextBox auslesen 682
245 ComboBox mit Autovervollständigung 685
246 Daten neben den Einträgen einer ListBox oder ComboBox verwalten 692
247 ListBox mit ToolTip für längere Einträge 693
248 ListView sortieren 695
249 Knoten einer Ebene in einem TreeView-Steuerelement vertauschen 698
250 Einzelne Knoten eines TreeView-Steuerelements sortieren 700
251 Die Werte einzelner Zellen in einem DataGridView setzen und lesen 702
252 Ein DataGridView anpassen 703
253 Ein DataGridView mit eigenen Spalten am Beispiel einer DateTimePicker-Spalte 707
254 Feiertage im MonthCalendar-Steuerelement darstellen 717
255 Drag&Drop von Dateien und Ordnern 720
256 Ein Ordner-Dialog 722
257 In einem Nicht-Tastatur-Ereignis herausfinden, ob eine bestimmte Taste betätigt ist 723
258 In einem eigenen Steuerelement verhindern, dass die Cursor-Tasten einen Fokuswechsel bewirken 725
259 Hooking – Eine MessageBox mit definierten Schalterbeschriftungen versehen 727

WPF 739

260 Fenster ohne Titelleiste 740
261 Den Handle eines WPF-Fensters ermitteln 740
262 Fenster über den Clientbereich verschiebbar machen 741
263 Windows-Nachrichten verarbeiten 741
264 Beim Maximieren eines Fensters ohne Titelleiste die Taskbar berücksichtigen 743
265 Fenster verlaufend füllen 747
266 Hintergrund mit Textur 748
267 Fenster mit speziellen Formen 751
268 Fenster mit dem Vista-Glas-Effekt ausstatten 753
269 Fenster in einer Schleife aktualisieren 757
270 Splash-Fenster 761
271 Die aktuelle DPI-Einstellung der Bildschirme des Systems ermitteln 765
272 Ein Fenster auf einem sekundären Bildschirm öffnen 766
273 Das Hauptfenster einer Anwendung ermitteln 768
274 Die absolute und die Bildschirm-Position eines Steuerelements ermitteln 768
275 Die optimale Position eines Fensters bezogen auf ein Steuerelement ermitteln 770
276 Beim Öffnen eines Fensters den Fokus setzen 774
277 Das Einfügen über die Zwischenablage abfangen 774
278 TextBox-Inhalt beim Eintritt komplett selektieren 775
279 TextBox auf Zahleingaben beschränken 776
280 Das TextChanged-Ereignis bei der ComboBox abfangen 781
281 Bei der Betätigung der Return-Taste die Tab-Taste simulieren 783
282 Drag&Drop von Dateien und Ordnern 785
283 In einem Nicht-Tastatur-Ereignis herausfinden, ob eine bestimmte Taste betätigt ist 786

LINQ und LINQ to SQL 789

284 Dynamische Abfragen 789
285 Ungleichheits-Verknüpfungen 793
286 Kreuzprodukt-Verknüpfungen 794
287 Kommaseparierte Dateien (CSV-Dateien) verarbeiten 794
288 Probleme mit der Benennung in LINQ-to-SQL-Modellen lösen 796
289 LINQ-to-SQL-Abfragen mit LIKE 798
290 SQL direkt ausführen 799
291 Die SQL-Anweisung einer LINQ-Abfrage evaluieren 800

Benutzer und Gruppen 805

292 Informationen zu den Benutzern eines Computers oder einer Domäne auflisten 805
293 Überprüfen, ob ein Benutzerkonto existiert 811
294 Benutzergruppen eines Computers oder einer Domäne auflisten 813
295 Benutzer ermitteln, die einer Gruppe angehören 817
296 Gruppen eines Benutzers abfragen 820
297 Eigenschaften eines Benutzers auslesen 823

298 Überprüfen, ob der aktuelle Benutzer einer in Windows vordefinierten Gruppe (z. B. Administratoren) angehört 830
299 Gruppen anlegen 833
300 Benutzer anlegen 835
301 Eigenschaften eines Benutzers ändern 837
302 Benutzer einer Gruppe zuweisen 839
303 Benutzer aus einer Gruppe entfernen 841
304 Benutzer löschen 842
305 Gruppe löschen 843
306 Den Namen des aktuellen Benutzers auslesen 844
307 Ein Programm unter einem spezifischen Benutzerkonto ausführen 845

Sicherheit 851

308 Strings sicher im Programm verwalten 851
309 Daten symmetrisch ver- und entschlüsseln 855
310 Asymmetrisches Verschlüsseln mit RSA 873
311 Sicherer Schlüsselaustausch mit ECDH 878
312 Daten mit Hashing-Verfahren verschlüsseln 879

Multimedia 891

313 Wave-Dateien abspielen 891
314 Multimedia-Dateien in einer Windows.Forms-Anwendung über MCI abspielen 892
315 Multimediadateien in einer WPF-Anwendung abspielen 908

Bildbearbeitung 911

316 Speicherschonend in einer Windows.Forms-Anwendung mit Bildern umgehen 911
317 Bitmap-Objekte aus BitmapSource-Objekten erzeugen 913
318 BitmapSource-Objekte aus Bitmap-Objekten erzeugen 914
319 Das Format eines Bilds auslesen 915
320 Bild-Metadaten auslesen 920
321 Das Aufnahmedatum eines Bilds auslesen 927
322 Eingelesene Bilder im Originalformat speichern 930
323 Bild in Byte-Array umwandeln 932
324 Byte-Array in Bild umwandeln 934
325 Bilder aus der Zwischenablage auslesen 936
326 Screenshot erstellen 938
327 Bilder skalieren 943
328 Thumbnails aus Bildern erzeugen 947
329 Bilder konvertieren 949
330 (JPEG-)Bilder mit definierter Qualität speichern 951
331 Bilder drehen, neigen und spiegeln 955
332 Bildausschnitte auslesen 960
333 Farben von Bildern auf andere Farben mappen 962
334 Farbinformationen von Bildern gezielt verändern 966
335 Ein Negativ eines Bilds erzeugen 971
336 Die einzelnen Pixel eines Bilds bearbeiten 973
337 Farb-Bilder in Graustufen-Bilder umwandeln 979

Zeichnen **983**

338 GDI-Probleme vermeiden 983
339 Einstellen der Grafik-Qualität unter GDI 984
340 Rechtecke mit abgerundeten Ecken zeichnen 987
341 Pfeile zeichnen 991
342 Transparente Bilder und Grafiken erzeugen 996
343 Bilder mit Schatten zeichnen 1001
344 Schräg zeichnen und Zeichenobjekte rotieren 1004
345 Den Drehpunkt eines Rechtecks so ermitteln, dass die Ecke links oben an derselben Position bleibt 1011
346 Text an einer definierten Position in 90-Grad-Schritten gedreht ausgeben 1013
347 Die Breite und Höhe eines auszugebenden Textes bestimmen 1015
348 Texte zentriert oder rechtsbündig zeichnen 1018
349 Strings beim Zeichnen wortgerecht umbrechen 1022

COM-Interop mit Office **1031**

350 Laufende COM-Komponenten-Instanzen referenzieren 1031
351 Word-Dokumente öffnen 1033
352 Word-Dokumente basierend auf einer Dokumentenvorlage erzeugen, füllen, ausdrucken und speichern 1037
353 Excel-Arbeitsmappen erzeugen 1041
354 Daten in Excel-Arbeitsmappen erweitern 1047
355 Performantes Lesen und Schreiben in Excel-Arbeitsmappen 1051
356 Kontakte aus Outlook auslesen 1053

Reflection und Serialisierung **1057**

357 Informationen über den Aufrufer in einer Methode oder einem Konstruktor ermitteln 1057
358 Programmcode dynamisch ausführen 1058
359 Assemblys dynamisch erzeugen 1061
360 Assemblys dynamisch laden 1062
361 Objekte über eine Datenvertrag-Serialisierung serialisieren 1066
362 Objekte binär serialisieren und deserialisieren 1071
363 Objekte nach XML serialisieren und von XML deserialisieren 1073
364 Font- und andere (serialisierbare) Objekte in einen String serialisieren 1076

Threading und asynchroner Methodenaufruf **1079**

365 In einem Thread sicher auf Steuerelemente zugreifen 1079
366 Easy-Threading mit der BackgroundWorker-Komponente 1082
367 Parameter an Threads übergeben und Ergebnisse auslesen 1086
368 Das ereignisbasierte asynchrone Entwurfsmuster implementieren 1092

Datenbank-Programmierung **1109**

369 Die Anzahl der Datensätze ermitteln, die eine Abfrage in einer Datenbank ergibt 1109
370 Datenbanken erzeugen 1110
371 Abfragen der automatisch vergebenen Id eines neuen Datensatzes 1113
372 Bilder und andere binäre Daten in einer Datenbank verwalten 1115

373 Backup und Restore einer SQL-Server-Datenbank über SMO 1120
374 Die verfügbaren SQL Server ermitteln 1124
375 Die Datenbanken einer SQL-Server-Instanz abfragen 1125

Teil III Anhang 1127

Änderungen gegenüber der zweiten Auflage 1129

Geänderte Rezepte 1129
Neue Rezepte 1133

Die ersten 255 Zeichen des Unicode-Zeichensatzes 1136

Stichwortverzeichnis 1139

Vorwort

Die Bibliothek des .NET Framework 3.5 enthält eine gewaltige Menge an Klassen, Strukturen und anderen Typen. Diese Vielzahl macht es trotz des hierarchischen Aufbaus dieser Bibliothek nicht immer einfach, die richtigen Typen für die Lösung eines Programmierproblems zu finden. Häufig ist aber auch der Einsatz der gefundenen Typen nicht trivial und erfordert einiges an Backgroundwissen. In einigen Fällen finden Sie in der .NET Framework Bibliothek aber einfach auch keine Klassen oder Strukturen, über die Sie Ihr Problem lösen könnten.

Hier setzt dieses Buch an. In 389[1] Rezepten finden Sie Lösungen zu Problemen, die in der Praxis immer wieder auftreten. Die meisten dieser Rezepte haben sich aus meiner jahrelangen Praxisarbeit mit .NET entwickelt. Andere entstanden aus Newsgroup-Beiträgen oder aus Fragen in meinen Seminaren und Workshops. Natürlich löst dieses Buch nicht alle Probleme. Die Praxisorientiertheit der Rezepte führt allerdings mit recht großer Wahrscheinlichkeit dazu, dass viele Ihrer Fragen zur C#-Programmierung in diesem Buch beantwortet werden.

Diese dritte Auflage des C# Codebook basiert natürlich auf der zweiten. Die Rezepte der vorherigen Auflage habe ich größtenteils beibehalten, aber komplett überarbeitet. Bei den meisten Rezepten betraf die Überarbeitung neue C#-Sprachfeatures (wie Lambda-Ausdrücke und Objekt-Initialisierer). Viele habe ich auch auf WPF erweitert (z. B. die Rezepte im Kapitel »Bildbearbeitung«). Bei vielen der alten Rezepte habe ich Verbesserungen einfließen lassen, die sich im Lauf der Zeit (auch durch die nette Mitwirkung einiger Leser) ergeben haben. Die Änderungen gegenüber der zweiten Auflage finden Sie übrigens im Anhang.

Das Buch enthält darüber hinaus auch neue Rezepte. Unter Ausreizung der zur Verfügung stehenden Seitenanzahl sind das immerhin 55. Dazu gehören neben einigen speziellen neuen Rezepten (wie endlich auch zwei, die die asymmetrische Verschlüsselung behandeln ☺) solche zu WPF, LINQ und LINQ to SQL. Als Zugabe finden Sie auf der Buch-CD auch einige umfangreiche Artikel, die diese neuen Technologien erläutern.

Einige Rezepte werden Ihnen u. U. sehr einfach erscheinen ☺. Diese beschreiben meist grundlegende Techniken, wie z. B. das Formatieren einer Zahl, die besonders von C#-Novizen des Öfteren nachgefragt werden. Diese Rezepte sollen aber eben genau die Zielgruppe der C#-Anfänger ansprechen. Die Vielzahl der Rezepte behandelt allerdings Probleme, deren Lösungen komplex sind und die auch von erfahreneren C#-Entwicklern nachgelesen werden müssen.

Bei der Erstellung der einzelnen Rezepte habe ich wesentlich mehr Wert auf Qualität als auf Quantität gelegt. Besonders wichtig war mir, dass Sie das, was dort programmiert ist, auch verstehen. Deshalb wird jedes Rezept erläutert, wobei ich auch auf die vielen möglichen Fallstricke eingehe. Außerdem enthält der entsprechende Quellcode jedes Rezepts Kommentare, die die Programmierung kurz erläutern. Daneben finden Sie keine »Quick and Dirty«-Lösung in diesem Buch. Ich habe jedes Rezept sehr intensiv getestet und potenzielle Fehlerquellen beseitigt (was manchmal für ein Rezept einige Stunden oder auch einen ganzen Tag in Anspruch genommen hat).

1. inklusive denen der Einführung in die verwendeten externen Technologien, die Sie auf der Buch-CD finden

Ich danke der Firma AUTOonline GmbH für die nette Unterstützung und das lange Warten darauf, dass ich mit meinen Büchern fertig werde. Während meiner (intensiven) Arbeit dort sind viele der neuen Rezepte entstanden und vorhandene gereift. Heiko Lehnert danke ich für das kritische Lesen der geänderten und neuen Rezepte. Durch seine Hilfe konnte ich bereits in der vorhergehenden Auflage einige Fehler beseitigen und Rezepte verständlicher gestalten. Der Frau meiner potenziellen Kinder, Tanja (Gliege) danke ich für ihre unendliche Geduld und ihre aufmunternden Worte. Axel Seibel, einem Leser der zweiten Auflage, danke ich für die Korrektur des Rezepts zum Auslesen der angezeigten Zeilen einer `TextBox`. Lutz Roeder danke ich für den hervorragenden .NET Reflector, ohne den das Recherchieren sehr schwer geworden wäre. Und schließlich danke ich den fünf Tibetern für die Kraft, die sie mir während der parallelen Arbeit an meinen aktuellen zwei Büchern gegeben haben.

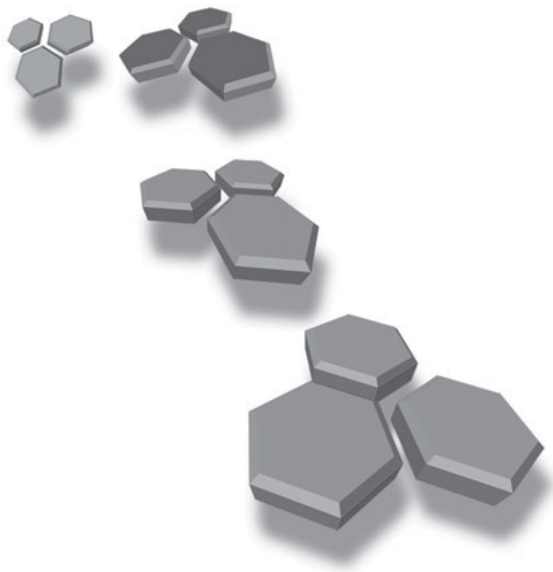
Falls Sie Anregungen oder Kritik oder einfach nur Lob zum Buch haben, freue ich mich über eine E-Mail.

Jürgen Bayer

codebook@juergen-bayer.net

Dublin

Teil I Einführung



Zum Buch

Der Inhalt

Das Buch beschreibt in 375 Rezepten (wenn das einführende Kapitel auf der CD mitzählt, sogar in 389) Lösungen zu Aufgabenstellungen, die in der Praxis häufig auftreten, aber meist nicht zu den trivialen Problemen gehören, die über eine einfache Verwendung einer .NET-Klasse gelöst werden können. Ausnahmen gibt es natürlich auch: Manchmal ist eine Lösung eigentlich trivial, aber bei vielen Programmierern unbekannt. Diese meist in Newsgroups gefragten Themen habe ich auch mit in den Rezeptteil aufgenommen.

Die meisten Rezepte dieses Buchs sind so allgemein, dass sie in jeder Anwendung einsetzbar sind. Der Fokus liegt aber auf der Programmierung von Windows-Anwendungen (über `Windows.Forms` und WPF). Techniken, die sich nur auf Webanwendungen beziehen, werden in diesem Codebook nicht beschrieben. Dazu gibt es ja schließlich das ASP.NET Codebook ☺. Damit Sie jetzt aber nichts Falsches denken: Nur die Kapitel »Windows.Forms« und »WPF« beziehen sich direkt auf die Programmierung von Windows.`Forms`- bzw. WPF-Anwendungen.

Einführende Artikel

Auf der Buch-CD finden Sie im Ordner *Zusatz-Artikel* einige teilweise umfangreiche Artikel. Der Artikel »Einführung in die unter .NET verwendeten externen Technologien« ist für das Codebook am wichtigsten. Er führt in die Arbeit mit dem Windows-API, mit COM, WMI und ADSI ein.

Die anderen Artikel führen unter .NET 3.5 (bzw. 3.0) neue Technologien ein. Dazu gehören LINQ, LINQ to SQL und WPF. Ich denke, wenn Sie sich mit diesen Themen noch nicht auskennen, loht es sich, die Artikel zu lesen.

Die Rezepte

Dieses Buch besteht neben der Einführung ausschließlich aus Rezepten. Jedes Rezept beschreibt die komplette Lösung eines Problems. Nur selten, wenn es sinnvoll erschien, habe ich ein Rezept mit einem anderen verknüpft.

Die meisten Rezepte setzen Klassen des .NET Frameworks 3.5 ein. Bei diesen Rezepten gehe ich davon aus, dass Sie mit den Grundlagen des .NET Frameworks und natürlich mit denen von C# vertraut sind. Im jeweiligen Rezept werden aber die verwendeten, besonderen Techniken, die nicht zu den Grundlagen gehören, erläutert und die benötigten Assembly-Referenzen und Namensraum-Importe aufgelistet. In den Rezepten habe ich aus Platz- und Übersichtsgründen darauf verzichtet, die Klassen voll (inklusive Namensraum) zu referenzieren. Deshalb gebe ich vor dem eigentlichen Quellcode an, welche Namensräume importiert werden müssen.

Einige Rezepte arbeiten mit externen Technologien wie dem Windows API, WMI, COM oder ADSI. Grundlagen zu diesen Technologien, die ich nicht voraussetzen kann, finden Sie in dieser Einführung. In den Rezepten setze ich dann voraus, dass Sie diese Grundlagen kennen.

Nur wenige Rezepte nutzen fertige Klassen oder Bibliotheken, die Sie frei aus dem Internet beziehen können (und meist auch auf der Buch-CD finden). Die in diesen Klassen implementierten Programme waren einfach zu komplex für das Buch. Und außerdem lag es mir fern, die teilweise genialen Klassen einfach zu kopieren. Ich wollte Ihnen aber eine Lösung für das jeweilige Problem anbieten. Bei den Rezepten, die mit externen Klassen arbeiten, habe ich natürlich darauf verzichtet, die Programmierung der eingesetzten Klassen zu erläutern, und zeige nur deren Anwendung.

Ein Rezept beschreibt kurz das Problem und den Lösungsansatz bzw. den Lösungsweg, meist ohne zu erläutern, in welchen Fällen Sie das Programm überhaupt benötigen. Sie werden schon wissen, warum Sie ein Rezept lesen ☺. Die meisten Rezepte liefern die Problemlösung in Form einer Methode. Viele dieser Methoden sind als Erweiterungsmethoden[1] in einer statischen Klasse definiert. Diese Methoden können Sie direkt auf den Typ anwenden, der am ersten Argument mit dem `this`-Modifizierer gekennzeichnet ist. Andere Methoden sind nur statisch, damit Sie diese einfach in eine separate Klasse einfügen und ohne eine Instanz zu bilden anwenden können. Wenn Sie Methoden aus Rezepten in eine Klasse einbinden wollen, die instanziert wird, und Sie keine statischen oder Erweiterungsmethoden wollen, entfernen Sie einfach gegebenenfalls das Schlüsselwort `this` vor dem ersten Argument und das Schlüsselwort `static`.

Die Problemlösung einiger Rezepte erforderte aber auch die Programmierung kompletter Klassen. Diese sind teilweise recht umfangreich, das Ergebnis kann sich aber sehen lassen. Ein gutes Beispiel ist die Klasse `Mci` im Rezept 314, die Ihnen ermöglicht, Multimedia-Dateien abzuspielen (wobei im Hintergrund das etwas schwierig anzuwendende MCI verwendet wird).

Für wenige Rezepte war es prinzipiell nicht möglich, Methoden oder Klassen zu entwickeln. Dabei handelt es sich meist um Beschreibungen von Vorgehensweisen wie z. B. die Erstellung eines Formulars mit einer speziellen Formgebung (Rezept 229 und 267). Die Quellcodes dieser Rezepte müssen Sie natürlich an Ihre Bedürfnisse bzw. Anforderungen anpassen.

Neben der Beschreibung der Problemlösung im Fließtext habe ich in den Listings immer auch Kommentare untergebracht. Teilweise führt das zu einer leichten Redundanz, da Dinge zweimal erläutert werden. Ich denke aber, dass ein Quelltext mit Kommentaren wesentlich besser zu verstehen ist als einer ohne. So haben Sie auch die Möglichkeit, den Fließtext schlicht zu ignorieren (was eigentlich nicht in meinem Sinne ist, denn wozu habe ich das Ganze dann geschrieben ☺), und können sich einfach direkt den Quelltext anschauen. Verstehen Sie diesen nicht, können Sie im Fließtext nachlesen.

Entwickelt habe ich die Rezepte unter Windows Vista mit .NET Framework 3.5. Getestet habe ich alle Rezepte unter Windows Vista und Windows XP.

Falls ein Rezept nur unter bestimmten Bedingungen funktioniert oder besondere Installationen erfordert, ist dies in der Regel im Rezept beschrieben. Bei den Rezepten, die WMI oder ADSI einsetzen, habe ich allerdings auch häufig darauf verzichtet, immer wieder zu schreiben,

1. Erweiterungsmethoden sind eine neues Feature des C#-3.0-Compiliers. Diese in einer statischen Klasse implementierten Methoden erweitern vorhandene Typen um neue Methoden. Das erste Argument einer Erweiterungsmethode muss dazu mit dem `this`-Schlüsselwort gekennzeichnet werden. Der Typ des ersten Arguments definiert den erweiterten Typen. Sobald eine Klasse mit Erweiterungsmethoden in ein Projekt eingebunden und der Namensraum importiert ist, stehen diese Methoden zur Verfügung.

dass WMI bzw. ADSI installiert sein müssen. Grundlagen dazu finden Sie ja auch hier in der Einführung.

Die Buch-CD

Die Buch-CD enthält die Beispiele des Buchs, ein Repository, in dem Sie sehr schnell in allen Rezepten suchen können, (teilweise umfangreiche) zusätzliche Artikel und Komponenten.

Falls Sie dieses Buch als PDF-Version gekauft haben, können Sie diese sehr wichtige CD bei Addison Wesley gegen eine Schutzgebühr nachbestellen. Auf der Buchseite bei Addison Wesley (siehe Link auf der Seite *www.juergen-bayer.net/buecher/csharpcodebook3*) finden Sie einen entsprechenden Link.

Das Codebook-Klassenbibliotheks-Projekt

Ich habe alle Rezepte, die in Methoden oder Klassen implementiert sind, in die Klassenbibliothek *Addison_Wesley.Codebook* eingebaut. Das entsprechende Projekt finden Sie auf der Buch-CD im Beispiele-Ordner. Wenn Sie wollen, können Sie diese Bibliothek einfach in Ihre Projekte einbinden. Nach dem Import des Namensraums `Addison_Wesley.Codebook` stehen Ihnen alle Erweiterungsmethoden und alle Klassen zur Verfügung. Die Namensräume der Codebook-Assembly entsprechen im Wesentlichen den einzelnen Kapiteln, der Name ist häufig aber abgekürzt und in englischer Schreibweise. Der Basis-Namensraum ist `Addison_Wesley.Codebook`. In diesem Namensraum sind die folgenden Unternamensräume enthalten:

- `Application`: Klassen für die Arbeit mit Anwendungen
- `Basics`: Klassen des Basics-Kapitels
- `Database`: Klassen für die Arbeit mit Datenbanken
- `Drawing`: Klassen für das Zeichnen
- `Files`: Klassen für die Arbeit mit Dateien
- `FileSystem`: Klassen für die Arbeit mit dem Dateisystem
- `Imaging`: Klassen für die Arbeit mit Bildern
- `Internet`: Klassen für den Zugriff auf das Internet
- `Multimedia`: Multimedia-Klassen
- `Security`: Klassen für sicherheitsrelevante Themen (Verschlüsseln etc.)
- `Serialisation`: Klassen für die Serialisierung
- `System`: Klassen für die Arbeit mit dem (Rechner-)System
- `Windows`: Klassen für die Arbeit mit Windows
- `Windows.Forms`: Klassen für die Arbeit mit `Windows.Forms`-Anwendungen
- `Windows.User`: Klassen für die Arbeit mit Windows-Benutzern und -Gruppen
- `WPF`: Klassen für die Arbeit mit WPF-Anwendungen

Die Klassen innerhalb der einzelnen Namensräume sind häufig in mehrere partielle Klassen aufgeteilt, wobei ein Teil jeweils einem Rezept entspricht. Der Grund dafür ist, dass diese Klassen in die Beispiele der entsprechenden Rezepte gelinkt sind.

Das Repository, die Beispiele und Komponenten

Auf der Buch-CD finden Sie ein Repository in HTML-Form, das die Programmcodes aller Rezepte enthält. Über dieses können Sie eine Problemlösung sehr schnell in Ihr Projekt einfügen. Die Rezepte im Repository korrespondieren natürlich mit denen im Buch.

Zudem finden Sie auf der CD auch Beispiele zu allen Rezepten und die in einigen Rezepten verwendeten Komponenten. In Beispielen, die Klassen verwenden, die auch in der Codebook-Klassenbibliothek vorkommen, sind die entsprechenden Dateien als Kopie enthalten, damit das jeweilige Beispiel für sich alleine funktioniert.

Unit-Testing

Da es bei einigen Rezepten Sinn macht, das Ergebnis der entwickelten Methoden zu testen, verwende ich dazu das Unit-Testing-Framework von Visual Studio. Die Test-Klassen habe ich in das separate Projekt *Addison_Wesley.Codebook.Tests* integriert, das in die Solution der Klassenbibliothek *Addison_Wesley.Codebook* integriert ist. Der (Basis-)Name einer Test-Klasse endet immer mit »Test«. Die Test-Klassen sind in partielle Klassen entsprechend den Rezepten aufgeteilt, für die sie verwendet werden.

Das Buch-Blog

Obwohl ich die einzelnen Rezepte sehr ausführlich unter Windows XP und Vista getestet habe, kann es in einem so umfangreichen Buch immer auch vorkommen, dass sich Fehler eingeschlichen haben. Einige Rezepte lassen sich eventuell auch verbessern oder funktionieren unter Umständen, unter denen ich diese nicht getestet habe, nicht oder nur eingeschränkt. An der Adresse *www.juergen-bayer.net/codebook* finden Sie deswegen einen Blog, in dem ich Verbesserungen und neue Rezepte veröffentlichen werde. Abonnieren Sie den RSS-Feed um auf dem Laufenden zu bleiben (ok, sooo oft werde ich nicht bloggen, aber ab und zu wird sich schon die eine oder andere Verbesserung ergeben).

Zum Füllen des Blog bin ich auch auf Ihre Mitarbeit angewiesen. Es wäre also sehr nett, wenn Sie mir – wie schon einige Leser für die vorhergehenden Auflagen – gefundene Fehler, Verbesserungen an Rezepten oder Anregungen per E-Mail an die Adresse *codebook@juergen-bayer.net* mitteilen.

Der Index

Jedes Fachbuch besitzt einen Index. Warum also darüber schreiben? Der Grund dafür ist, dass Sie im Index teilweise Problemlösungen finden, die nicht im Inhaltsverzeichnis auftauchen. Das liegt daran, dass ich innerhalb einiger Rezepte so ganz nebenbei auch andere Probleme lösen musste, für die ich keine eigenen Rezepte vorgesehen hatte. Damit Sie auch diese finden, habe ich die Lösungen immer mit in den Index aufgenommen.

Voraussetzungen

Das Buch setzt voraus, dass Sie die Grundlagen von C# 3.0 beherrschen. Neben den Grundlagen der Sprache (Variablen, Schleifen, Verzweigungen etc.) sollten Sie die Idee der .NET-Klassenbibliothek kennen, mit Namensräumen umgehen können, wissen, was eine Klasse ist, und die wichtigen grundlegenden Klassen wie `String`, `Convert` oder `DateTime` kennen. Sie sollten außerdem eine Entwicklungsumgebung wie Visual Studio 2008 (Express) oder SharpDevelop (*www.icsharpcode.net/OpenSource/SD/Default.aspx*) einsetzen.

Typografische Konventionen und Icons

Das Buch verwendet einige typografische Konventionen und Icons, die hier kurz erläutert werden.

- Im Fließtext werden Wörter, die zu einem Programm gehören, in der Schriftart `Courier` ausgegeben,
- *kursiv formatierte* Wörter bezeichnen Dateinamen, Ordnernamen und allgemeine Namen,
- Internetadressen werden folgendermaßen formatiert: *www.addison-wesley.de*,
- WÖRTER IN KAPITÄLCHEN stehen für Begriffe, die Teil einer Programmoberfläche sind (wie z. B. die Beschriftung eines Schalters oder ein Link auf einer Internetseite).

Hinweis

In einem solchen Kasten werden Hinweise und besondere Informationen gekennzeichnet, die Sie beachten sollten.

Achtung

Dieser Kasten kennzeichnet Hinweise, die Sie auf jeden Fall beachten sollten, da deren Nichtbeachtung zu Fehlern oder Problemen führen könnte.

Tipp

Der Tipp-Kasten beinhaltet Tipps (was auch sonst), z. B. Hinweise auf Webseiten, die weitere Informationen zum jeweiligen Thema anbieten.

Exkurs

Ein solcher Kasten beinhaltet einen Exkurs.

Recherche-Möglichkeiten

Die Programmierung mit derart komplexen Programmierumgebungen wie dem .NET Framework und den zusätzlich eingesetzten Technologien wie dem Windows-API, WMI und ADSI ist ohne die Möglichkeit einer Recherche kaum möglich. Besonders im Internet finden Sie gerade für .NET eine Vielzahl an Informationsquellen, die Sie nutzen können, um Ihre Probleme effizienter zu lösen.

Das vorliegende Buch hilft Ihnen (hoffentlich ☺) bei der Lösung der meisten Aufgaben. Da Sie aber wahrscheinlich auch einmal spezielle Probleme oder solche lösen müssen, die im Buch nicht beschrieben werden, zeige ich Ihnen die Möglichkeiten der Recherche, die ich im Laufe meiner Tätigkeit als Softwareentwickler (im .NET-Bereich) gefunden habe.

Wichtige .NET-Websites

Viele Websites im Internet beschäftigen sich mit .NET. Ich kenne natürlich nicht alle, während meiner Arbeit mit .NET habe ich aber einige sehr gute Websites entdeckt, die ich hier kurz vorstelle. Auf den meisten dieser Websites finden Sie neben Beispiel-Quellcodes auch kompakte Artikel, die spezifische und teilweise komplexe Themen (meist) verständlich erläutern.

Hinweis

Die Links zu den Websites, die ich hier beschreibe, finden Sie auch in der Datei *Links.html* auf der Buch-CD.

WindowsClient.net

Diese zu Microsoft gehörende Seite (*windowsclient.net*) stellt eine Vielzahl an Artikeln, Beispielen, Foren etc. für die Entwicklung von Windows-Anwendungen zur Verfügung. Viele Artikel stammen aus dem MSDN, sind aber hier übersichtlich zusammengefasst.

ASP.net

Die ebenfalls zu Microsoft gehörende Website ASP.net (*www.asp.net*) liefert eine Menge allgemeine und spezifische Informationen zu der Web-Programmierung mit ASP.NET.

Codezone

Die deutschsprachige Codezone (*www.codezone.de*) ist ein »Developer Knowledge Network rund um Microsoft Technologien«. Hier finden Sie eine große Anzahl an Ressourcen zu .NET-Themen. Die hervorragende Idee von Codezone ist, dass dort jeder Entwickler seine Ressourcen veröffentlichen kann. Eine ebenfalls gute Idee ist, dass die Einträge bei Codezone neben einer kurzen Beschreibung lediglich einen Link auf die entsprechenden Webseiten oder den Download beinhalten. Auf diese Weise ist die Veröffentlichung von Artikeln oder anderen Ressourcen in der jeweiligen Community sehr einfach (und wird z. B. von mir genutzt ☺).

dotnetjunkies

Bei den *dotnetjunkies* (*www.dotnetjunkies.com*) finden Sie viele, gute und hervorragend formatierte How-To-Artikel, Tutorial-Artikel und Beispiel-Codes, hauptsächlich zu ASP.NET und dessen Umfeld.

The Code Project

Das *Code Project* (*www.codeproject.com*) veröffentlicht auf seiner Website im Vergleich zu anderen Websites zwar nicht allzu viele Artikel. Diese sind dafür aber qualitativ hochwertig und behandeln auch spezielle Themen.

Auf der Seite *www.codeproject.com/KB/dotnet* finden Sie allgemeine .NET-Artikel. Die Seite *www.codeproject.com/KB/cs* verweist auf Artikel zu C#. Andere Seiten, die Sie über das Menü der Einstiegsseite erreichen, beschäftigen sich (u. a.) mit speziellen .NET- und C#-Themen. Daneben finden Sie aber auch Seiten zu wichtigen Windows-Technologien wie z. B. DirectX und dem Windows-API.

.NET 247

.NET 247 (*www.dotnet247.com*) versteht sich als »erste unabhängige .NET-Programmierer-Referenz« und verweist in gut geordneten Kategorien auf Artikel anderer Websites zu wichtigen .NET-Referenz- und -Programmier-Themen. Neben diesen finden Sie auf der Startseite Links zu den wichtigsten Namensräumen des .NET Framework und auf der jeweiligen Seite die im Namensraum enthaltenen Klassen. Die Seite einer .NET-Klasse enthält wiederum Links zu Ressourcen, Artikeln, Beispielen, Diskussionen etc., die mit der Klasse in Zusammenhang stehen. Das Ganze ist sehr hilfreich, wenn Sie nicht wissen, wie eine Klasse in der Praxis angewendet wird.

C# Corner

Die »C#-Ecke« (*www.c-sharpcorner.com*) enthält sehr viele Artikel, Quellcodes und Links zu C#-Themen. Die einfache Navigation und die Unterteilung der Inhalte in gut sortierte Kategorien machen diese Website für .NET-Programmierer sehr wertvoll. Die Artikel sind meist kurz gefasst, enthalten aber alle wesentlichen Informationen (was nicht bei allen .NET-Seiten der Fall ist).

C# Help

Auf der Website von *C# Help* (*www.csharphelp.com*) finden Sie viele Artikel zu C#, die sich teilweise mit Grundlagen, aber auch mit fortgeschrittenen Themen beschäftigen. Leider existierte zur Zeit der Drucklegung dieses Buchs kein Verzeichnis aller Artikel (nur die aktuellen waren auf der Startseite verlinkt). Sie können aber natürlich nach archivierten Artikeln suchen.

.NETheute

.NETheute (*www.dotnetheute.com*) ist eine hervorragende deutschsprachige Website mit vielen Artikeln zu .NET und dem Umfeld.

dotnet-snippets.de

Auf der deutschen Seite *dotnet-snippets.de* veröffentlichen .NET-Programmierer einzelne Codeschnipsel zur Lösung aller möglichen Probleme. Die Seite bietet die Möglichkeit, die einzelnen Codeschnipsel zu bewerten, so dass Sie erkennen können, was andere darüber denken. Sie ist außerdem gut organisiert, bietet die Möglichkeit zu suchen und einen RSS-Feed, über den Sie über die neuen Codeschnipsel informiert werden.

devX

Der .NET-Bereich der umfangreichen *devX*-Website (*www.devx.com/dotnet*) enthält viele Artikel zu .NET, die teilweise frei verfügbar sind, teilweise aber (leider) zum kostenpflichtigen »Premier Content« gehören. Die vielen und guten Artikel von devX sind leider nicht allzu einfach zu finden (klicken Sie auf den Link MORE C# ARTICLES, um eine Liste anzuzeigen). Sie können aber natürlich (leider mit einer eingeschränkten Google-Suche) auch nach Artikeln suchen. Der E-Mail-Newsletter, den Sie über die .NET-Startseite abonnieren können, ist hingegen hervorragend. Er enthält eine Kurzbeschreibung der neuesten Artikel und ermöglicht deren direkten Aufruf über den E-Mail-Client.

ONDotnet.com

Auf der .NET Website des Verlags O'Reilly (*www.ondotnet.com*) finden Sie eine Vielzahl an interessanten Artikeln, die teilweise Grundlagen-, aber auch spezifische Themen behandeln.

Lutz Roeder's PROGRAMMING.NET

Lutz Roeder veröffentlicht im .NET-Bereich seiner Website (*www.aisto.com/roeder/dotnet*) einige sehr interessante .NET-Komponenten wie z. B. einen »Reflector«, über den Sie .NET-Komponenten und Assemblys erforschen können, und eine CommandBar-Komponente, über die Sie verschiedene Befehls-Symbolleisten in Ihre `Windows.Forms`-Anwendung integrieren können.

George Shepherd's Windows Forms FAQ

Auf der Seite von *SyncFusion*, einem Hersteller von .NET-Komponenten, veröffentlicht George Shepherd ein umfangreiches FAQ, das sich nicht nur mit den `Windows.Forms`-Komponenten, sondern auch mit Themen wie GDI+, dem Windows-API, COM und allgemeinen Framework-Tipps beschäftigt. Die im FAQ enthaltenen Tipps und Tricks wurden dabei verschiedenen Newsgroups und Mailing-Listen entnommen bzw. stammen von Mitarbeitern von SyncFusion. Sie finden dieses FAQ an der Adresse *www.syncfusion.com/faq/winforms*.

Newsgroup-Recherche bei Google

Die bekannte Suchmaschine Google ermöglicht nicht nur die Suche nach Websites. Google archiviert daneben auch alle wichtigen Newsgroups und ermöglicht die Suche in diesem Archiv. Gerade .NET-Themen werden in einer Vielzahl spezialisierter Newsgroups umfangreich diskutiert. Hier finden Sie in den meisten Fällen Lösungen zu Ihren Problemen oder wenigstens unbeantwortete Fragen, an denen Sie erkennen, dass auch andere Programmierer ähnliche Probleme haben. Sie können natürlich einzelne Newsgroups in einem Newsgroup-Reader (wie Outlook oder dem Newsgroup-Reader des Mozilla-Browsers) abonnieren und die einzelnen Beiträge durchgehen. Da aber gerade die englischsprachigen Newsgroups mit Beiträgen geradezu »bombardiert« werden, ist diese Art der Suche sehr ineffizient. Die Google-Suche bringt eigentlich immer wesentlich schneller ein Ergebnis. Ehrlich gesagt hätte ich dieses Buch kaum ohne diese hervorragende Möglichkeit, in den aktuellen und älteren Newsgroup-Beiträgen zu suchen, schreiben können ☺.

Sie können die (erweiterte) Google-Newsgroup-Suche direkt über die Adresse *groups.google.de/advanced_search* öffnen. Auf der Suchseite können Sie nach verschiedenen Kriterien suchen. Für die Praxis wichtig sind aber lediglich die Eingabe der Suchkriterien und die Einschränkung der Newsgroups. Abbildung 1 zeigt eine Suche nach einem »folder dialog« in allen Newsgroups, die »dotnet« im Namen tragen.

Wenn Sie nach englischen Begriffen suchen, finden Sie wesentlich mehr (englischsprachige) Beiträge, als wenn Sie deutsche Begriffe einsetzen. Die deutschen Newsgroups werden meist wesentlich weniger frequentiert als die englischen.

Die Einschränkung auf passenden Newsgroups im Feld NEWSGROUP ist sehr wichtig, da Sie ansonsten auch Beiträge aus Newsgroups finden, die nichts mit .NET oder C# zu tun haben. Das Einschränken ist auch sehr einfach, da (nach meiner Erfahrung) alle deutsch- oder englischsprachigen .NET-Newsgroups den Text »dotnet« im Namen tragen. Die Angabe von **dotnet** im Feld NEWSGROUP teilt Google dann auch mit, in allen Newsgroups zu suchen, deren Name »dotnet« enthält. Eine spezielle Einschränkung auf C# (über **csharp**) ist meiner Ansicht nach nicht notwendig, da Sie ohne Probleme auch Lösungen für Visual Basic .NET und mit ein wenig Gehirnschmalz auch Lösungen für Visual C++ .NET nach C# umsetzen können. Außerdem behandeln einige Newsgroups relativ sprachunabhängig allgemeine .NET-Themen und tragen folglich kein »csharp« im Namen. Es schadet allerdings auch nicht, einfach beide Newsgroup-Muster anzugeben: **dotnet*; *csharp**.

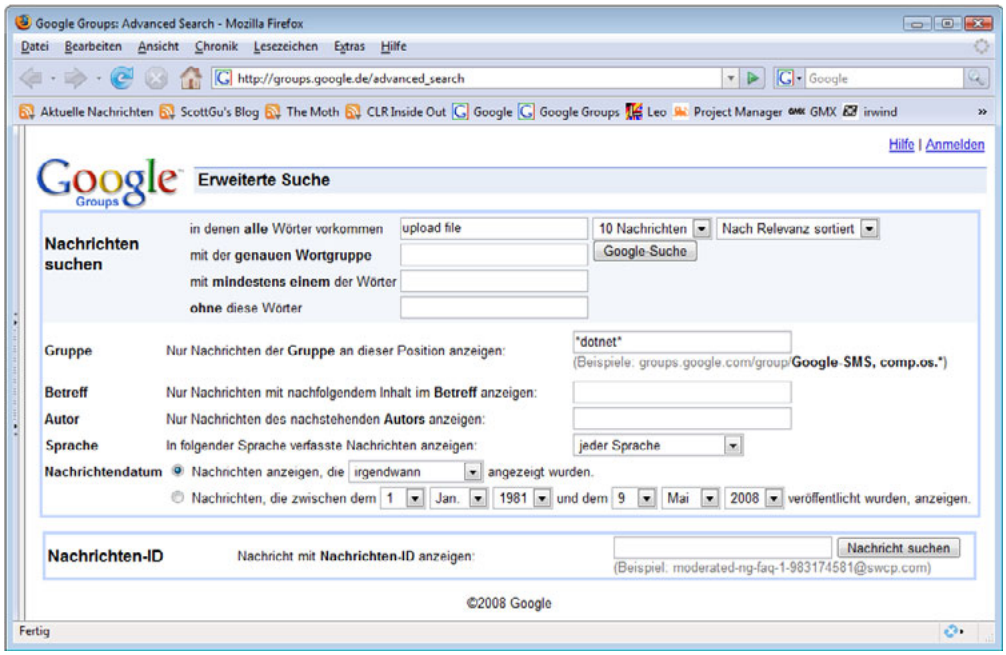


Abbildung 1: Google-Newsgroup-Recherche in allen .NET-Newsgroups

Google listet nach der Suche die gefundenen Beiträge sortiert nach Relevanz auf (Abbildung 2).

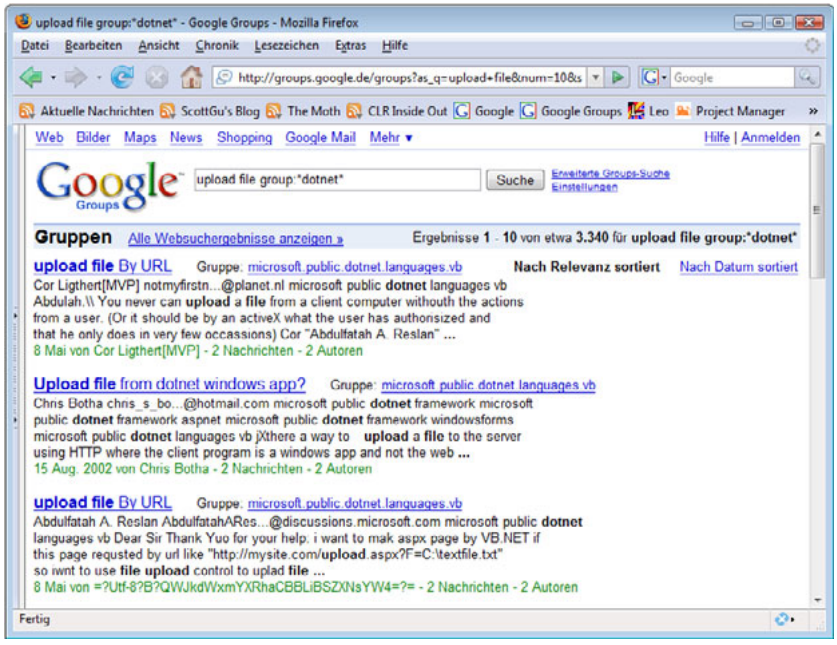


Abbildung 2: Das Ergebnis einer Google-Newsgroup-Recherche nach den Suchbegriffen »upload file« in den .NET-Newsgroups

Die Links führen zu den jeweiligen Beiträgen, die teilweise ursprüngliche Beiträge und oft Antworten auf vorhergehende Beiträge sind (was Sie am »RE:« erkennen).

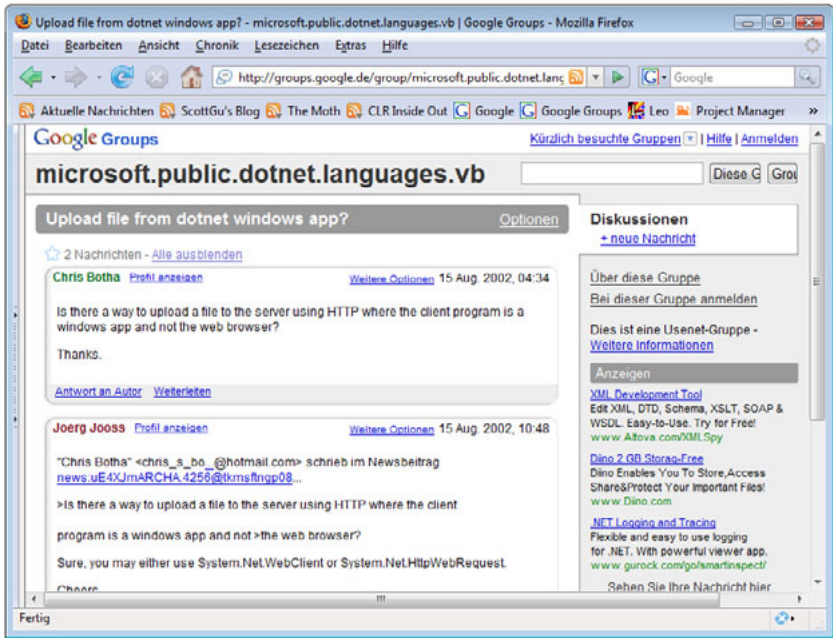


Abbildung 3: Gefundene Newsgroup-Artikel

In diesem Beispiel (das Sie aufgrund der permanent neuen Beiträge in den Newsgroups wahrscheinlich so nicht nachvollziehen können) führt die Suche direkt zum Erfolg. Das ist zwar nicht immer so, manchmal müssen Sie einige Beiträge durchforsten oder auch einmal die Suchbegriffe ändern um eine Lösung zu finden. Wenn Sie diese allerdings gefunden haben, erspart dies meist eine Menge Such- und Probierarbeit.

Suche bei Microsoft

Microsoft stellt eine sehr große Anzahl an Informationen in seinem MDSN (Microsoft Developer Network) und in der Knowledge Base zur Verfügung. Eine kombinierte Suche ist (u. a.) über die Seite *search.msdn.microsoft.com/search* möglich, die Sie bei Bedarf auf Deutsch umstellen können. Hier können Sie auch mit deutschen Begriffen suchen. Achten Sie darauf, dass Sie im Suchbegriff immer ».NET« oder »C#« angeben, damit Sie nur Artikel finden, die etwas mit C# oder .NET zu tun haben.

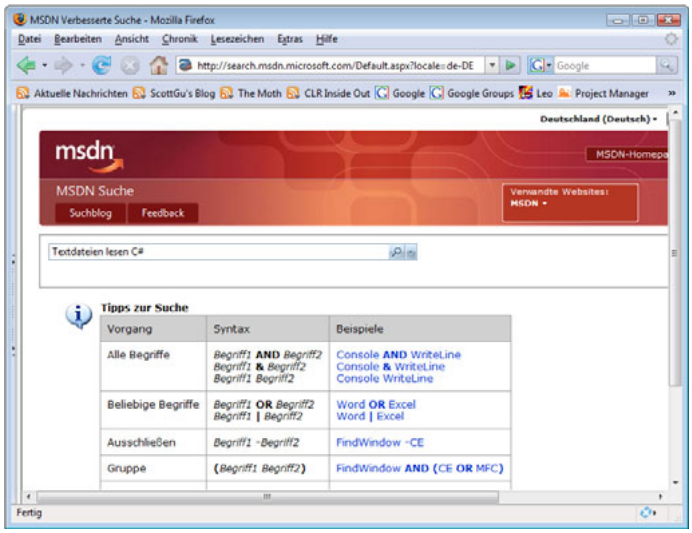


Abbildung 4: Suche in der Knowledge Base und im MSDN von Microsoft



Abbildung 5: Ergebnis der MSDN-Suche von Microsoft

Eine andere Möglichkeit ist die Suche auf der englischsprachigen Microsoft-Seite, die nach meinen Erfahrungen häufig andere bzw. bessere Ergebnisse liefert. Diese Suche erreichen Sie über den Link *search.microsoft.com/advancedsearch.aspx?mkt=en-US*. Geben Sie als Suchseiten die Knowledge Base und das MSDN an.

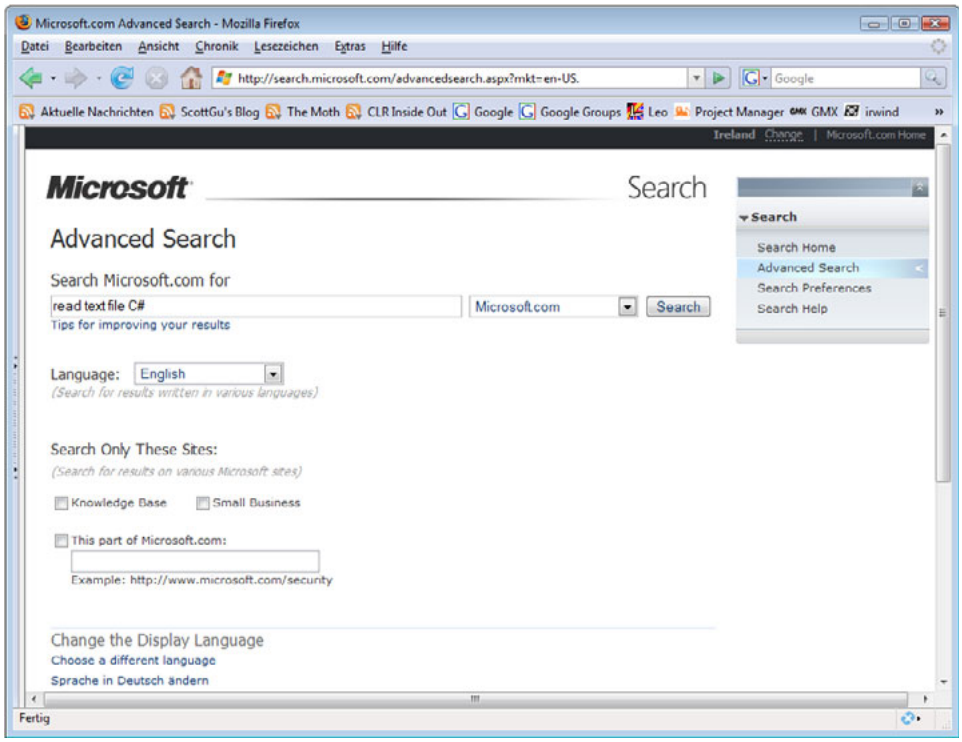


Abbildung 6: Suche in der englischsprachigen Microsoft-Knowledge Base und im MSDN

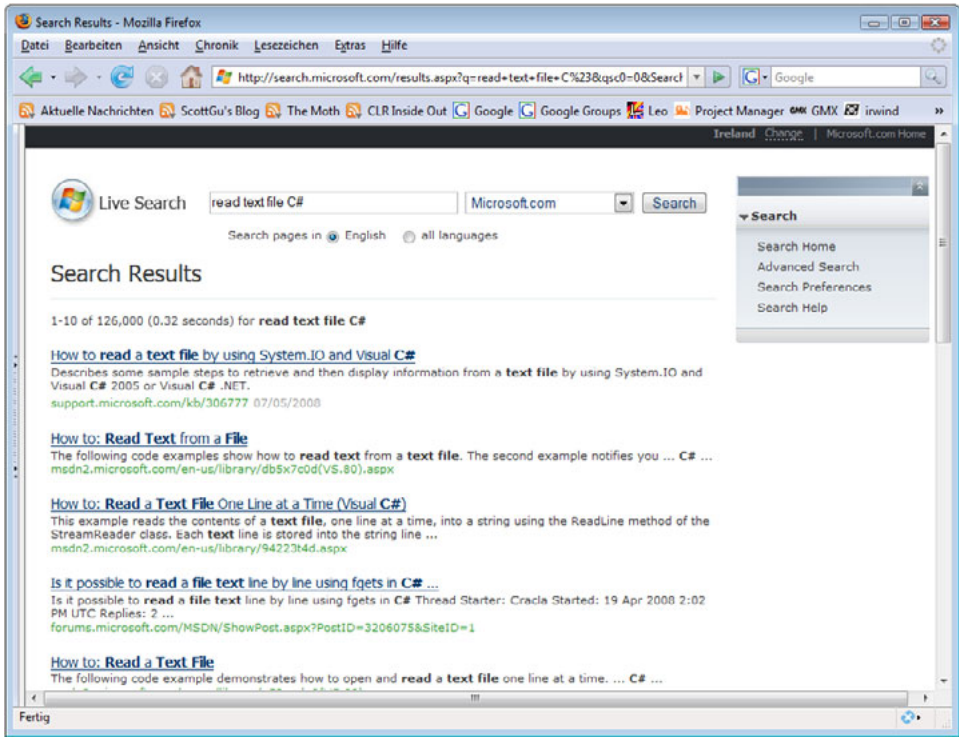


Abbildung 7: Ergebnis der Knowledge Base-/MSDN-Suche bei Microsoft

Hinweis

Microsoft verwirrt allerdings mit weiteren Suchseiten, wie z. B. der Seite *www.microsoft.com/germany/msdn/search.aspx*, die im »MSDN-Portal« sucht. Probieren Sie die Microsoft-Suchen aus, Sie werden sehen, dass die Ergebnisse der einzelnen Suchseiten recht unterschiedlich sind. Die Links zu den Suchseiten finden Sie in der Datei *Links.html* auf der Buch-CD.

Debugging des .NET-Framework-Quellcodes

Der Quellcode des .NET Framework liefert so manche Hinweise zur Lösung verschiedener Probleme. Zum Disassemblieren des Quellcodes können Sie den .NET Reflector von Lutz Roeder verwenden, den ich im nächsten Abschnitt vorstelle. Sie können aber Visual Studio 2008 (leider nicht in den Express-Editionen) auch so einstellen, dass Sie in den Quellcode des .NET Framework hinein debuggen können. Die dazu erforderlichen Debugsymbole und den Quellcode erhalten Sie von einem Symbol-Server bei Microsoft. Microsoft stellt den Quellcode unter einer speziellen Lizenz zur Verfügung, der Sie im Verlauf der Installation zustimmen müssen. Diese Lizenz besagt im Wesentlichen, dass Sie den Quellcode zu Evaluierungszwecken verwenden können, aber nicht in Ihren Programmen einsetzen dürfen.

Zum Debuggen des .NET-Framework-Quellcodes sollten Sie ggf. zuerst ein Hotfix installieren, das einen Fehler in Visual Studio 2008 beseitigt. Dieser Fehler bewirkt, dass Visual Studio beim Debuggen mit Symbolen, die von einem externen Symbol-Server kommen, sehr langsam

ist. Das Hotfix wird im Service Pack 1 enthalten sein. Wenn Sie also das Service Pack 1 installiert haben, müssen Sie das Hotfix nicht installieren. Sie finden das Hotfix auf der Microsoft-Seite für das »Visual Studio and .NET Framework Hotfix Public Availability Program« (*connect.microsoft.com/VisualStudio/content/content.aspx?ContentID=3705*) im Abschnitt VISUAL STUDIO 2008 unter der Knowledgebase-Nummer *KB944899*. Beachten Sie, dass Sie das Hotfix in der Sprache herunterladen sollten, in der Sie Visual Studio installiert haben. Zurzeit des Schreibens dieser Zeilen waren die Hotfixes allerdings nur in englischer Sprache verfügbar. Das englische Hotfix 944899 konnte ich mit der ersten deutschen Version von Visual Studio leider nicht installieren.

Nun müssen Sie in den Visual-Studio-Optionen im Register DEBUGGING / ALLGEMEIN die Option NUR EIGENEN CODE AKTIVIEREN (NUR VERWALTET) aus- und die Option QUELLSERVERUNTERSTÜTZUNG AKTIVIEREN einschalten (Abbildung 8).

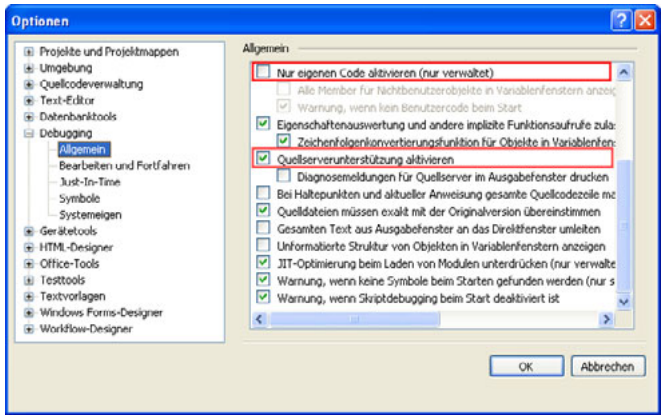


Abbildung 8: Die Basis-Optionen zur Verwendung der Microsoft-Debugsymbole für den .NET-Framework-Quellcode

Die Option NUR EIGENEN CODE AKTIVIEREN (NUR VERWALTET) können Sie später wieder ausschalten, wenn Sie verhindern wollen, dass Visual Studio beim Debuggen in Quellcode hineinspringt, der nicht zu der aktuellen Projektmappe gehört.

Nun müssen Sie noch den Symbolserver eintragen, was im Register SYMBOLE möglich ist. Klicken Sie dazu im rechten Bereich auf den linken Schalter und geben Sie in der Liste den Server *http://referencesource.microsoft.com/symbols* ein. Unten sollten Sie dann noch einen Ordner angeben, in dem die Symbole, die vom Symbolserver heruntergeladen werden, zwischengespeichert werden, damit diese nicht jedes Mal neu abgefragt werden müssen (Abbildung 9).

Die Option SPEICHERORTE NUR DURCHSUCHEN, WENN SYMBOLE MANUELL GELADEN WERDEN bewirkt, dass Visual Studio die Debugsymbole nur dann lädt, wenn Sie diese manuell laden. Ist die Option eingeschaltet, müssen Sie beim Debuggen in der Aufrufliste den Befehl SYMBOLE LADEN wählen, um die Symbole der betreffenden Assembly zu laden. Ist diese Option nicht eingeschaltet, werden die kompletten verfügbaren Symbole beim Öffnen eines Projekts oder, wenn Sie die Option SYMBOLE BEIM SCHLIEßEN DES DIALOGFELDS MIT AKTUALISIERTEN EINSTELLUNGEN LADEN eingeschaltet lassen, beim Schließen des Optionen-Dialogs automatisch geladen. Beim

ersten Laden kann das natürlich einige Zeit in Anspruch nehmen, da die Debugsymbole ja zunächst vom Symbol-Server heruntergeladen werden müssen. Das nächste Laden ist dann viel schneller, weil die Symbole lokal zur Verfügung stehen. Der Vorteil des Ausschaltens der Option SPEICHERORTE NUR DURCHSUCHEN, WENN SYMBOLE MANUELL GELADEN WERDEN ist, dass Sie beim Debuggen nicht jedes Mal die Symbole erst manuell laden müssen. Der Nachteil ist, dass die geladenen Symbole relativ viel Platz im Arbeitsspeicher beanspruchen. Shawn Burke, der Microsoft-Entwickler, der dieses Feature zur Verfügung gestellt hat, empfiehlt in seinem Blog, diese Option einzuschalten (*blogs.msdn.com/sburke/archive/2008/01/16/configuring-visual-studio-to-debug-net-framework-source-code.aspx*). Mit eingeschalteter Option können Sie beim Debuggen entscheiden, ob Sie die Symbole laden und in den .NET-Framework-Quellcode hineinspringen.

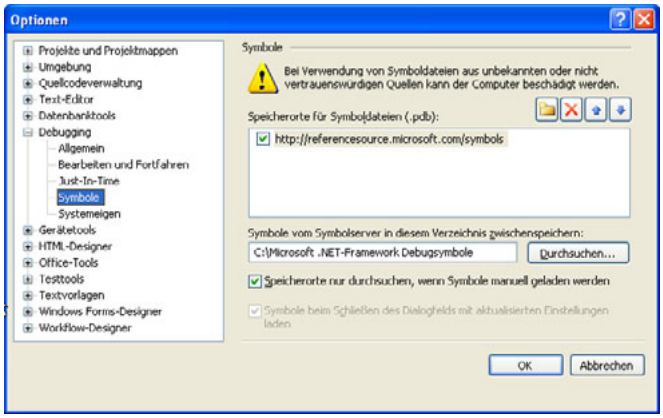


Abbildung 9: Angabe des Symbolservers und des Cache-Ordners zur Verwendung der Microsoft-Debugsymbole für den .NET-Framework-Quellcode

Wenn Sie die Optionen dann mit OK schließen, erscheint eine Warnmeldung, die aussagt, dass Sie Symbole nur von vertrauenswürdigen Quellen beziehen sollen. Wenn Sie Microsoft für vertrauenswürdig halten, bestätigen Sie diese Meldung mit JA.

Wenn Sie nun in einem Projekt einen Haltepunkt setzen, das Programm starten und dieses an dem Haltepunkt anhält, sehen Sie in der Aufrufliste schon einmal wesentlich mehr Informationen als zuvor. Die einzelnen Sprungpunkte der Liste sind aber noch ausgegraut, weil zurzeit noch keine Symbole geladen sind. Ich verwende als Beispiel eine WPF-Anwendung mit einem Fenster mit einem Schalter, in dessen `Click`-Ereignisse ich den Titel des Fensters ändere. Um die Symbole zu laden, die Ihnen erlauben in den Quellcode zu springen, wählen Sie den Befehl SYMBOLE LADEN im Kontextmenü eines der Einträge in der Aufrufliste, der zu der entsprechenden Assembly gehört. Um in die `Title`-Eigenschaft eines WPF-Fensters zu springen, wäre das die Assembly *PresentationFramework.dll*.

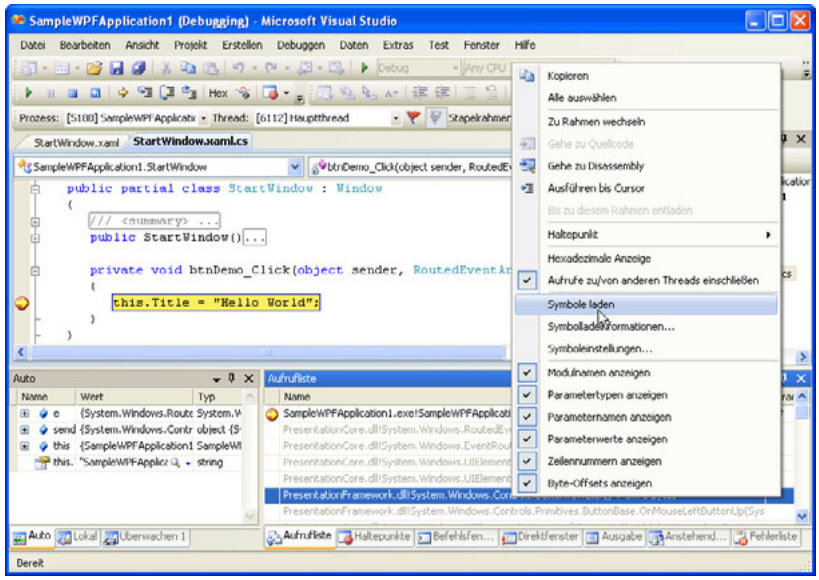


Abbildung 10: Laden von Symbolen für die Assembly PresentationFramework.dll

Beim ersten Laden von Symbolen müssen Sie die Microsoft-Benutzungs-Bestimmungen für die Symbole akzeptieren. Wenn Sie das gemacht haben, dauert das Laden ggf. eine Weile. Danach können Sie über die Aufrufliste und mit F11 in den .NET-Framework-Quellcode springen. Beim ersten Springen in den Quellcode des .NET Framework müssen Sie noch die Microsoft-Linzenzbestimmungen akzeptieren. Lesen Sie diese gut durch und bestätigen Sie den Dialog mit ACCEPT, wenn Sie den Bestimmungen zustimmen. Danach steht der Debugger im Quellcode des .NET Framework und Sie können diesen weiter durchgehen (was natürlich auch von den geladenen Symbolen abhängt). Was sehr interessant ist, ist die Tatsache, dass Sie den gesamten Quellcode erhalten, inklusive den Original-Kommentaren.

Die Symbole und der Quellcode werden in dem Ordner gespeichert, den Sie in den Optionen als Cache-Ordner angegeben haben. Beim nächsten Laden von Symbolen werden diese wesentlich schneller geladen, da sie aus dem Cache-Ordner gelesen werden können.

Sie können sogar im Quellcode des .NET Framework Haltepunkte setzen. Dazu müssen Sie die entsprechende Datei natürlich zunächst in Visual Studio laden. Entweder setzen Sie dazu einen Haltepunkt in Ihrem Quellcode an einem Feature, das in der gewünschten Klasse des .NET Framework enthalten ist, und Sie springen in diese Klasse, oder Sie öffnen eine bereits in Ihrem Cache-Ordner enthaltene Datei (in einem der Unterordner des Ordners *src*). Wenn Sie den Haltepunkt dann gesetzt haben und das Programm starten, meldet Visual Studio allerdings, dass der Haltepunkt zurzeit nicht erreicht wird, weil der Quellcode Unterschiede zur Originalversion aufweist. Der Grund dafür ist, dass der vom Microsoft Symbolserver heruntergeladene Quellcode im oberen Bereich einen Copyright-Abschnitt enthält, der in den Quellcode-Dateien, mit denen die Debugsymbole erzeugt wurden, nicht enthalten war. Um Haltepunkte in den .NET-Framework-Quellcode-Dateien zu ermöglichen müssen Sie die Debugging-Option QUELLDATEIEN MÜSSEN EXAKT MIT DER ORIGINALVERSION ÜBEREINSTIMMEN abschalten (Abbildung 11).

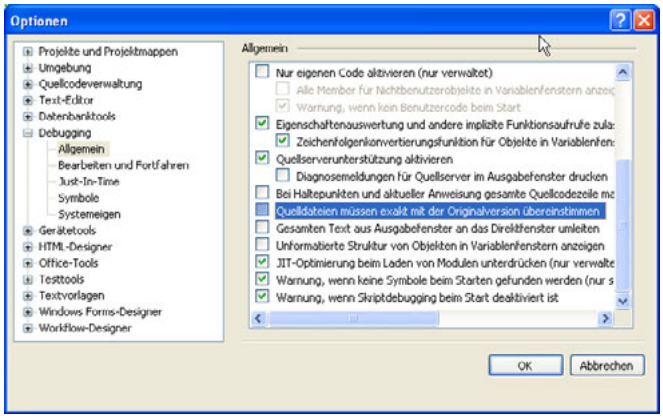


Abbildung 11: Abschalten der Option »Quelldateien müssen exakt mit der Originalversion übereinstimmen« um Haltepunkte im Quellcode des .NET Framework zu ermöglichen

Diese Option sollten Sie nach dem Debuggen im .NET-Framework-Quellcode wieder einschalten.

Der .NET Reflector von Lutz Roeder

Der .NET Reflector von Lutz Roeder ist mittlerweile in .NET-Kreisen ein beliebtes Werkzeug zum Erforschen von Klassenbibliotheken. .NET Reflector ermöglicht es auf einfache Weise alle Typen einer Klassenbibliothek zu erforschen und zeigt sogar den disassemblierten Quellcode von Eigenschaften oder Methoden an. Besonders die ungeschützte Bibliothek des .NET Framework liefert so manche Erkenntnisse. Bei der Entwicklung von eigenen Steuerelementen ist es z. B. oft hilfreich, sich anzuschauen, wie Microsoft bestimmte Probleme in den Standard-Steuerelementen gelöst hat.

Den .NET Reflector finden Sie an der Adresse *www.aisto.com/roeder/dotnet*. Die Installation ist .NET-typisch einfach: Kopieren Sie die erhaltene *exe*-Datei in einen beliebigen Ordner. Beim ersten Aufruf müssen Sie die Version des .NET Framework auswählen, deren wichtige Assemblys der .NET Reflector automatisch in seine Assembly-Liste einträgt. Über den Menübefehl File / Open können Sie beliebige Assemblys nachladen, über File / Open Cache können Sie Assemblys aus dem GAC nachladen. Der Menübefehl File / Open List ermöglicht die Definition eigener Listen mit den Assemblys der verschiedenen .NET-Framework-Versionen.

Im linken Fensterbereich wählen Sie eine Klasse aus und erforschen deren Member. Zum Disassemblieren eines Members klicken Sie einfach doppelt auf den Eintrag oder Sie betätigen die Leertaste. Der .NET Reflector zeigt dann im rechten Bereich den disassemblierten Quellcode (Abbildung 12).

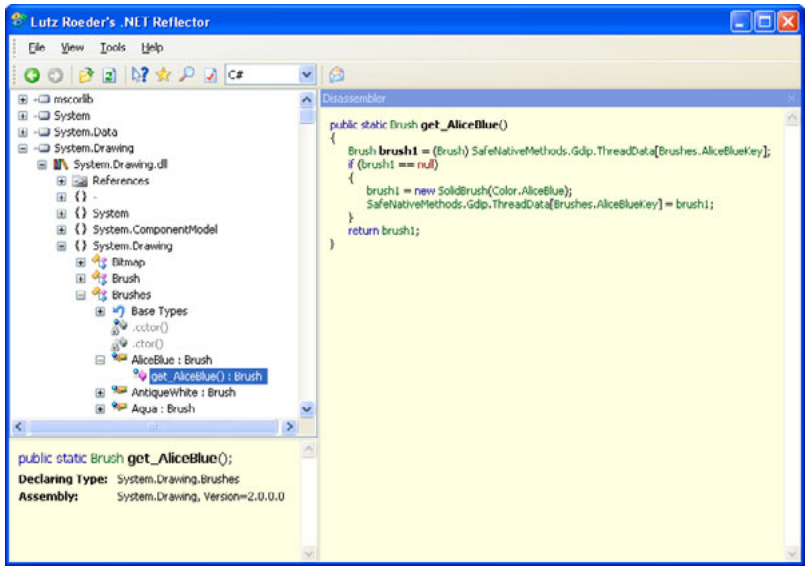
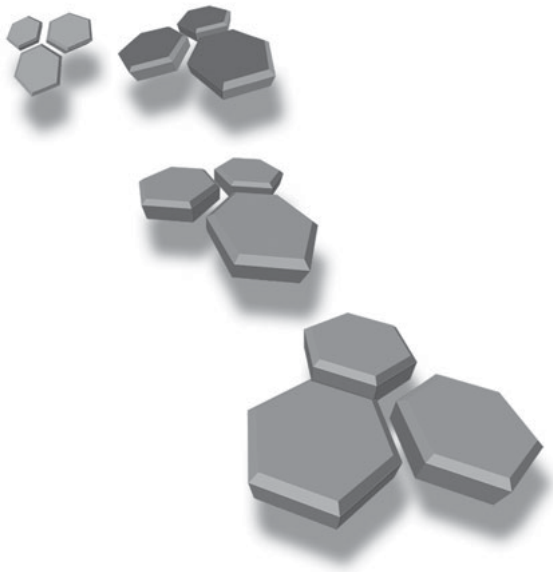


Abbildung 12: Der .NET Reflector von Lutz Roeder zeigt den Quellcode der get-Methode der AliceBlue-Eigenschaft der Brushes-Klasse an

Ein weiteres interessantes Feature sind Add-Ins, die Sie in den .NET Reflector integrieren können. Diese finden Sie auf der Seite *www.codeplex.com/reflectoraddins*. Ein sehr hilfreiches Add-In ist der *File Disassembler* (*www.denisbauer.com/NETTools/FileDisassembler.aspx*), über den Sie für einen einzelnen Typen oder sogar für eine komplette Assembly C#-Quellcode-Dateien erzeugen können. Add-Ins integrieren Sie über das Menü VIEW / ADD-INS in den Reflector.

Daneben bietet der Reflector noch weitere Features wie z. B. die Möglichkeit, nach Typen oder Membern zu suchen und diese zu analysieren (was u. a. aussagt, wo die Typen oder Member verwendet werden). Probieren Sie diese Features einfach aus.

Teil II Rezepte



Basics

Datum und Zeit

Anwendungen, Anwendungs-Konfiguration, Prozesse und Dienste

Dateisystem

Text-, binäre und Zip-Dateien

XML

System

Internet

Windows.Forms

WPF

LINQ und LINQ to SQL

Benutzer und Gruppen

Sicherheit

Multimedia

Bildbearbeitung

Zeichnen

COM-Interop mit Office

Reflection und Serialisierung

Threading und asynchroner Methodenaufruf

Datenbank-Programmierung

Basics

1 Zahlen formatieren

Dies ist eines der einfachsten Rezepte in diesem Buch, und wahrscheinlich werden Sie bereits wissen, wie Sie Zahlen formatiert ausgeben. Aber in meinen C#-Einführungs-Seminaren ist diese Frage eine der meistgestellten ☺.

Zahlen können Sie ganz einfach über die `ToString`-Methode eines Zahl-Datentyps formatieren. Dieser Methode können Sie einen String übergeben, der ein Format-Muster enthält. Das Muster »0.00« steht zum Beispiel dafür, dass die Zahl mit mindestens einer Vorkomma- und zwei Nachkommastellen dargestellt wird:

```
double number = 0.1599;
string formattedNumber = number.ToString("0.00");
```

In diesem Beispiel resultiert auf einem deutschen System der String »0,16«. Beim Formatieren auf eine angegebene Anzahl Nachkommastellen wird die Zahl gerundet, wobei das bei uns übliche kaufmännische Rundungsverfahren angewendet wird (siehe Rezept 8).

Alternativ können Sie Zahlen auch über die statische `Format`-Methode der `String`-Klasse formatieren. Dieser Methode übergeben Sie einen Format-String, der aus beliebigen Zeichen und Platzhaltern bestehen kann. Ein Platzhalter wird in geschweifte Klammern eingefügt und mit einem Index versehen. Der erste Index (0) steht für den ersten Wert, den Sie am zweiten (`params`-)Argument übergeben, der zweite Index steht für den zweiten Wert etc.

Ein einfacher Platzhalter sieht folgendermaßen aus: »{0}«. Hinter dem Platzhalter-Index können Sie durch einen Doppelpunkt getrennt ein Formatmuster angeben. So können Sie eine Zahl zum Beispiel auch auf zwei Stellen hinter dem Komma formatieren:

```
double number = 0.1599;
string formattedNumber = String.Format("{0:0.00}", number);
```

Tabelle 1 beschreibt die wichtigsten Formatierzeichenfolgen.

Formatier-zeichenfolge	Bedeutung	Beispielwert	Formatierter Wert
C	Währung mit Standard-Dezimal-stellen	100.955	”100,96 €”
C*n*	Währung mit definierten Dezi-malstellen. *n* gibt die Dezimal-stellen an.	100.955 (mit *C4* formatiert)	”100,9550 €”
E	Exponential-Format	1235	”1,235000E+003”
F	Festkommazahl mit im System eingestellten Dezimalstellen und ohne Tausendertrennzeichen	1234.955	”1234,96”

Tabelle 1: Die Formatierzeichenfolgen für Zahlwerte

Formatier-zeichenfolge	Bedeutung	Beispielwert	Formatierter Wert
F*n*	Festkommazahl mit definierten Dezimalstellen und ohne Tausendertrennzeichen. *n* gibt die Dezimalstellen an.	1234.955 (mit *F4* formatiert)	”1234,9550”
G	Allgemeine Zahl. Die Zahl wird abhängig von ihrem Typ und ihrer Größe möglichst kompakt formatiert. Bei großen Zahlen wird das Exponentialformat verwendet. Dieses Format entspricht der Standardformatierung der `ToString`-Methode, wenn diese ohne Formatierzeichen aufgerufen wird.	1234567890123456 1234.5678	”1,23456789012346E+15” ”1234.5678”
G*n*	Allgemeine Zahl mit definierter Anzahl Stellen	1234567890123456 (mit *G4* formatiert)	”1,235E+15”
N	Standard-Zahl mit im System eingestellten Dezimalstellen und Tausendertrennzeichen	1234.955	”1.234,96”
N*n*	Standard-Zahl mit definierten Dezimalstellen und ohne Tausendertrennzeichen. *n* gibt die Dezimalstellen an.	1234.955 (mit *N1* formatiert)	”1.235,0”
P	Prozentwert (die Zahl wird durch 100 geteilt und es wird ein Prozentzeichen angehängt). Für die Anzahl der Dezimalstellen wird die System-Voreinstellung verwendet.	0.1234	”12,34%”
P*n*	Prozentwert mit definierter Anzahl Nachkommastellen. *n* gibt die Dezimalstellen an.	0.1234 (mit *P1* formatiert)	”12,3%”
X	Hexadezimales Format	255	”FF”
X*n*	Hexadezimales Format mit angegebener Anzahl Stellen	255 (mit *X4* formatiert)	”00FF”
0.00	Benutzerdefiniert: Mindestens eine Vorkomma- und genau zwei Nachkommastellen.	0.345 12.345 1234.567	”0,35” ”12,35” ”1234,57”
.00	Benutzerdefiniert: Genau zwei Nachkommastellen	0.345 12.345 1234.567	”,35” ”12,35” ”1234,57”

Tabelle 1: Die Formatierzeichenfolgen für Zahlwerte (Forts.)

Formatier-zeichenfolge	Bedeutung	Beispielwert	Formatierter Wert
#,#	Benutzerdefiniert: Mit Tausendertrennzeichen und ohne Nachkommastellen (diese werden aufgerundet)	1234.567	"1.235"
#,#0.00	Benutzerdefiniert: Mit Tausendertrennzeichen, mindestens einer Vorkommastelle und zwei Nachkommastellen	1234.567	"1.234,57"

Tabelle 1: Die Formatierzeichenfolgen für Zahlwerte (Forts.)

Für die benutzerdefinierten Formatierzeichen (0 und #) zeigt Tabelle 1 nur Beispiele. # steht für eine oder keine Ziffer, 0 für genau eine Ziffer. Sie können diese Symbole natürlich beliebig kombinieren. In der Tabelle habe ich lediglich die gängigsten Formatierungen aufgelistet.

2 Negative und positive Zahlen unterschiedlich formatieren

In vielen Finanz-Anwendungen ist zu sehen, dass negative Zahlen anders formatiert (z. B. in Klammern) ausgegeben werden als positive. Um dies zu erreichen setzen Sie die `ToString`-Methode des Zahltyps oder die `Format`-Methode der `String`-Klasse ein, deren Grundlagen ich bereits in Rezept 1 erläutert habe. Der »Trick« ist, dass Sie die Formatzeichenfolge nun zwei- oder dreiteilen, wobei Sie die einzelnen Teile durch Semikolons trennen. Der erste Teil steht für positive Werte, der zweite für negative und der dritte für 0. Da Sie nun explizit positive von negativen Werten unterscheiden, werden negative Zahlen nicht mehr automatisch mit einem vorangestellten Minuszeichen formatiert.

Das folgende Beispiel formatiert eine `double`-Zahl so, dass positive Zahlen normal mit führender Null und zwei Nachkommastellen angezeigt werden. Negative Zahlen werden ähnlich formatiert, aber in Klammern ausgegeben:

```
double number = 123.456;
string formattedNumber = number.ToString("0.00;(0.00)");
```

Listing 1: Unterschiedliches Formatieren einer Zahl je nach Vorzeichen

Das Resultat dieser Formatierung ist für 123.456 "123,46" und für -123.456 "(123,46)"

3 (Ganz-)Zahlen hexadezimal darstellen

Das hexadezimale Zahlenformat bietet besonders für Programmierer den wichtigen Vorteil, dass ein Byte mit zwei Ziffern dargestellt werden kann (00 bis FF). Es eignet sich deshalb hervorragend zur optischen Auswertung der einzelnen Bytes einer Zahl. Das ist wohl auch der Hauptgrund dafür, dass Technologien wie HTML für Farbangaben neben vordefinierten Konstanten auch hexadezimal dargestellte Zahlwerte erlauben, die sich aus drei Bytes zusammen-

setzen. Das linke Byte steht dabei für den Rotanteil der Farbe, das mittlere für den Grünanteil und das rechte für den Blauanteil. Die Farbe FFA000 steht z. B. für einen Orangeton.

Eine Ganzzahl können Sie sehr einfach entweder über die `ToString`-Methode eines Integer-Datentyps oder über die `Format`-Methode der `String`-Klasse in einen hexadezimalen Wert konvertieren, indem Sie als Format-Bezeichner ein *X* übergeben:

```
int i = 0xFAFBFC;
string hexNumber = String.Format("{0:X}", i);
```

Listing 2: Formatieren einer Ganzzahl als Hexadezimalwert

4 Hexadezimale Zahlen in dezimale Zahlen konvertieren

Hexadezimal dargestellte Zahlwerte können Sie über die Methoden `ToByte`, `ToSByte`, `ToInt16`, `ToUInt16`, `ToInt32`, `ToUInt32`, `ToInt64` und `ToUInt64` der `Convert`-Klasse in eine Integer-Zahl konvertieren. Am ersten Argument übergeben Sie dazu den zu konvertierenden String, am zweiten die Basis der Zahl, die in diesem Fall 16 ist. Da die Konvertierung auch fehlschlagen kann, sollten Sie die dann erzeugte Ausnahme abfangen:

```
string hexNumber = "FFA033";

// Konvertieren des Hex-Werts in eine long-Zahl
long number = 0;
try
{
   number = Convert.ToInt64(hexNumber, 16);
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Konvertieren: {0}", ex.Message);
}
```

Listing 3: Konvertieren eines hexadezimalen Werts in einen long-Wert

Alternativ zur Ausnahmebehandlung können Sie auch vor der Konvertierung abfragen, ob der zu konvertierende String eine gültige Zahl speichert. Das Rezept 16 nimmt Ihnen diese Aufgabe ab.

5 Ganzzahlen dual darstellen

In einigen Fällen ist es hilfreich oder einfach nur informativ, eine Zahl in ihrer dualen Darstellung vor Augen zu haben. Sinn macht das allerdings nur bei ganzen Zahlen, da Fließkommazahlen auf eine recht komplexe Weise im Speicher abgelegt werden (als Vorzeichenbit, Mantisse und Exponent).

Das .NET Framework bietet dazu keine Möglichkeit. Sie müssen die Formatierung selbst programmieren. Dazu können Sie für positive Zahlen ganz einfach in einer Schleife die einzelnen Bits der Zahl durchgehen. Wenn Sie auf die Zahl ein bitweises Und mit 1 anwenden, können Sie den Zustand des jeweils rechten Bits ermitteln. In der Schleife shiften[1] Sie die Zahl dann um ein Bit nach rechts, damit beim nächsten Durchlauf das nächste Bit verglichen wird. Das jeweilige Ergebnis hängen Sie einfach an eine String-Variable an.

Für negative Zahlen sieht das Ganze etwas anders aus: Diese werden nämlich in der Regel im so genannten *Zweierkomplement* (siehe *de.wikipedia.org/wiki/Zweierkomplement*) dargestellt. In diesem werden positive Zahlen mit einer führenden Null dargestellt und weiter nicht verändert. Negative Zahlen werden mit einer führenden Eins dargestellt. Die einzelnen binären Ziffern des Absolutwerts der negativen Zahl werden negiert und zum Ergebnis wird 1 aufaddiert. Das ergibt dann automatisch eine führende Eins und gleichzeitig eine etwas ungewohnte Darstellung. -4 wird dual z. B. als 11111100 dargestellt, -128 als 10000000. Zur Verdeutlichung folgen hier die einzelnen Rechenschritte am Beispiel:

- Den Absolutwert in Dualsystem umrechnen: $-4_{10} = 00000100_2$
- Invertieren: 11111011_2
- Eins aufaddieren: $11111011_2 + 0000\ 0001_2 = 11111100_2$

Da es nur Sinn macht positive und negative Zahlen auf die gleiche Weise darzustellen, habe ich die Methoden zur Umwandlung von Zahlen in das Dualsystem mit einem Argument ausgestattet, über das Sie angeben können, ob das Ergebnis »normal« (mit führendem Plus- und Minuszeichen) oder im Zweierkomplement dargestellt werden soll.

Für eine `int`-Zahl sieht das Ganze dann so aus wie in der Methode `GetDualNumber` in Listing 4.

```
public static string GetDualNumber(this int number, bool useTwosComplement)
{
   // Variable für das Ergebnis
   string result = "";

   // Absolutwert der Zahl berechnen und ermitteln,
   // ob diese negativ ist
   uint absoluteNumber = (uint)(number < 0 ? number * -1 : number);
   bool isNegative = (number < 0);

   // Die Bits 0 bis 32 durchgehen
   for (int i = 0; i < 32; i++)
   {
```

Listing 4: Erweiterungsmethode zur Konvertierung einer int-Zahl in eine Dualzahl

1. Beim Shiften über die Operatoren << (Linksshift) und >> (Rechtsshift) werden die einzelnen Bits einer Zahl um ein Bit nach links bzw. rechts verschoben. Beim Linksshiften fällt das linke Bit »heraus«, rechts wird ein 0-Bit eingefügt. Beim Rechtsshiften fällt das rechte Bit heraus und links wird ein 0-Bit eingefügt.

```
      // Über ein bitweises Or ermitteln, ob das Bit gesetzt ist,
      // und das Ergebnis vorne an die Stringvariable anfügen
      result = (absoluteNumber & 1).ToString() + result;

      // Die Bits um eine Position nach rechts shiften
      absoluteNumber >>= 1;
   }

   if (useTwosComplement)
   {
      // Zweierkomplement verwenden
      if (isNegative)
      {
         // Das Ergebnis negieren
         string temp = result;
         result = null;
         foreach (char digit in temp)
         {
            if (digit == '0')
               result += '1';
            else
               result += '0';
         }

         // 1 aufaddieren
         temp = result;
         result = null;
         bool invert = true;
         for (int i = temp.Length - 1; i >= 0; i--)
         {
            if (temp[i] == '0')
            {
               if (invert)
               {
                  result = '1' + result;
                  invert = false;
               }
               else
               {
                  result = '0' + result;
               }
            }
            else
            {
               result = (invert ? '0' : '1') + result;
```

Listing 4: Erweiterungsmethode zur Konvertierung einer int-Zahl in eine Dualzahl (Forts.)

```
            }
         }
      }
   }
   else
   {
      // Kein Zweierkomplement verwenden
      if (isNegative)
         result = "-" + result;
      else
         result = "+" + result;
   }

   // Ergebnis zurückgeben
   return result;
}
```

Listing 4: Erweiterungsmethode zur Konvertierung einer int-Zahl in eine Dualzahl (Forts.)

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` importieren.

Ich habe natürlich Methoden für alle Ganzzahltypen implementiert, die ich hier allerdings – bis auf die Variante, der ein `uint`-Wert übergeben wird – nicht vorstelle. Sie finden Methoden für alle Ganzzahltypen im Repository und im Beispiel zu diesem Rezept.

Ganzzahlen ohne Vorzeichen müssen etwas anders behandelt werden. Hier muss sichergestellt werden, dass die Zahl im Zweierkomplement dargestellt werden kann, sofern dieses verwendet werden soll. Zur Erinnerung: Positive Zahlen werden im Zweierkomplement mit einer Null im äußerst linken Bit dargestellt. Das linke Bit darf also nicht von der Zahl selbst bereits verwendet werden. Bei Zahltypen mit Vorzeichen ist dies bereits durch den Typ selbst sichergestellt (Ganzzahlen mit Vorzeichen werden im Zweierkomplement gespeichert).

Deshalb überprüfen die Varianten von `GetDualNumber`, denen jeweils eine Ganzzahl ohne Vorzeichen übergeben wird, ob diese kleiner/gleich der Maximalzahl des äquivalenten vorzeichenlosen Typs (Ganzzahl-)geteilt durch 2 ist. Bei `sbyte` kommt dabei 127 heraus (255 / 2), was dem positiven Maximalwert von `byte` entspricht. Daneben entfällt bei diesen Methoden die Umrechnung negativer Zahlen in das Zweierkomplement.

```
public static string GetDualNumber(this uint number, bool useTwosComplement)
{
   // Variable für das Ergebnis
   string result = "";
   // Sicherstellen, dass die Zahl für das Zweierkomplement
   // klein genug ist, das letzte (linke) Bit darf nicht
   // von der Zahl selbst verwendet werden
   if (useTwosComplement)
   {
      uint maxNumber = uint.MaxValue / 2;
      if (number > maxNumber)
      {
         throw new ArgumentException("Wenn die Dualzahl im Zweierkomplement" +
            "dargestellt werden soll, darf die Zahl nicht größer sein als " +
            maxNumber);
      }
   }

   // Die Bits 0 bis 32 durchgehen
   for (int i = 0; i < 32; i++)
   {
      // Über ein bitweises Or ermitteln, ob das Bit gesetzt ist,
      // und das Ergebnis vorne an die Stringvariable anfügen
      result = (number & 1).ToString() + result;

      // Die Bits um eine Position nach rechts shiften
      number >>= 1;
   }

   if (useTwosComplement == false)
   {
      // Kein Zweierkomplement verwenden
      result = "+" + result;
   }

   // Ergebnis zurückgeben
   return result;
}
```

Listing 5: Erweiterungsmethode zur Konvertierung einer uint-Zahl in eine Dualzahl

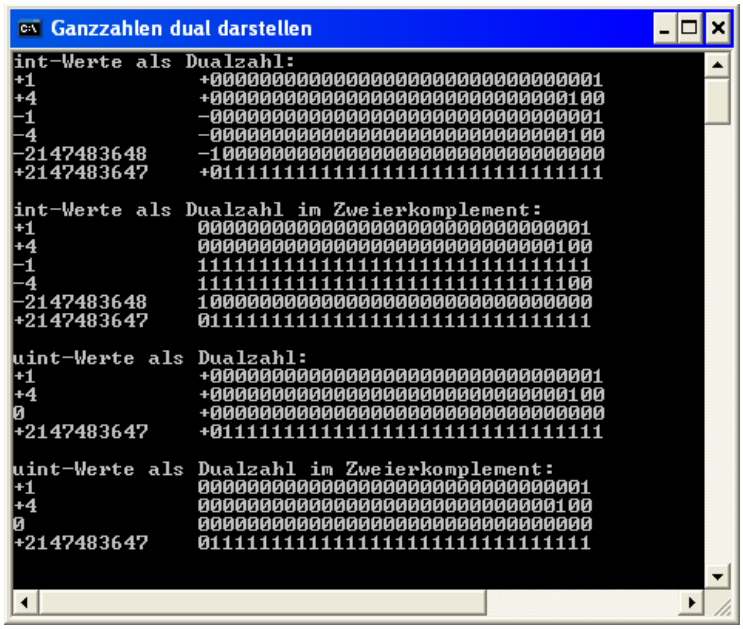


Abbildung 13: Das Testprogramm in Aktion

Die kleinste Zahl eines Integertyps mit Vorzeichen entspricht übrigens dem Wert 0 mit Vorzeichenbit (weswegen der Absolutwert des kleinsten Werts eines Integer-Typs mit Vorzeichen um 1 größer ist als der Absolutwert des größten Werts). Deswegen gibt das Programm für den kleinsten `int`-Wert (-2147483648) den Dualwert -10000000000000000000000000000000 aus.

6 Dualzahlen in Ganzzahlen konvertieren

Ähnlich wie hexadezimal dargestellte Zahlen (vgl. Rezept 4) können Sie auch dual dargestellte Zahlen in einen Integer-Wert konvertieren. Dazu verwenden Sie dieselben Methoden der `Convert`-Klasse wie im genannten Rezept, geben als Basis aber nun 2 an:

```
string dualNumber = "00001010";

// Konvertieren des Dualwerts in eine byte-Zahl
byte number = 0;
try
{
   number = Convert.ToByte(dualNumber, 2);
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Konvertieren: {0}", ex.Message);
}
```

Listing 6: Konvertieren einer Dualzahl in eine dezimale Zahl

Bei Konvertierungen mit den Methoden `ToByte`, `ToUint16`, `ToUint32` und `ToUint64` wird der String »normal« ausgewertet. Da diese Methoden keine negativen Werte erlauben, darf der Dualstring nicht mit einem Minuszeichen beginnen (ein Plus-Zeichen ist allerdings erlaubt).

Verwenden Sie die Methoden `ToSByte`, `ToInt16`, `ToInt32` und `ToInt64`, wird der String allerdings entsprechend dem Zweierkomplement (siehe Rezept 5) ausgewertet. Der String "11111100" ergibt in diesem Fall die Zahl -4. Der Dualwert muss hier (natürlich) exakt der Bitgröße des jeweiligen Datentyps entsprechen.

7 Zahlen in verschiedenen Notationen darstellen

Zur Formatierung einer Zahl in verschiedene Notationen können Sie spezielle Formatzeichen einsetzen, die Sie z. B. der `ToString`-Methode eines numerischen Datentyps übergeben oder im Formatstring der `Format`-Methode der `String`-Klasse einsetzen können. Das .NET Framework definiert dazu die folgenden Zeichen:

- *C*: Formatierung als Währung,
- *e* oder *E*: Formatierung im wissenschaftlichen Format in der Form *Mantisse*E*Exponent*,
- *P*: Formatierung als Prozentwert,
- *G*: Formatierung als normale Zahl ohne Tausendertrennzeichen,
- *F*: Formatierung mit mindestens einer Ziffer links vom Dezimaltrennzeichen und zwei Dezimalziffern,
- *N*: Formatierung als normale Zahl mit Tausendertrennzeichen und zwei Dezimalziffern,
- *H*: Hexadezimale Formatierung.

Listing 7 zeigt einige Beispiele für die Formatierung in verschiedenen Notationen. Zum Kompilieren dieses Beispiels müssen Sie den Namensraum `System` importieren.

```
// Codierung der Konsole umstellen, damit das Euro-Zeichen korrekt ausgegeben
// werden kann
Console.OutputEncoding = Encoding.UTF8;

// Währung
Console.WriteLine(doubleNumber.ToString("C"));

// Wissenschaftliches Format
Console.WriteLine(doubleNumber.ToString("e"));
Console.WriteLine(doubleNumber.ToString("E"));
```

Listing 7: Darstellung von Zahlen in verschiedenen Notationen

```
// Formatierung als Prozentwert
Console.WriteLine(doubleNumber.ToString("P"));

// Formatierung als normale Zahl ohne Tausendertrennzeichen
Console.WriteLine(doubleNumber.ToString("G"));

// Formatierung mit mindestens einer Ziffer links
// und zwei Ziffern rechts vom Dezimaltrennzeichen
Console.WriteLine(doubleNumber.ToString("F"));

// Formatierung als normale Zahl mit Tausendertrennzeichen
// und zwei Dezimalziffern
Console.WriteLine(doubleNumber.ToString("N"));

// Hexadezimal (nur Integer-Werte)
Console.WriteLine(intNumber.ToString("X"));
```

Listing 7: Darstellung von Zahlen in verschiedenen Notationen (Forts.)

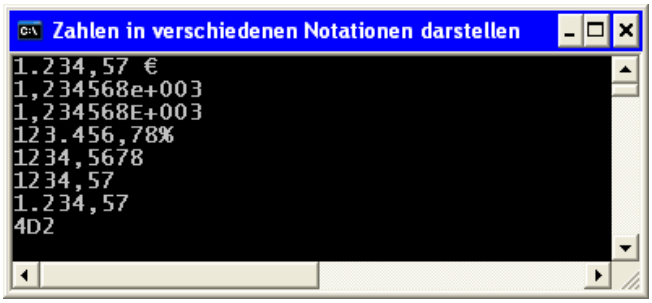


Abbildung 14: Das Beispielprogramm in Aktion

Hinweis

Die Konsole verwendet normalerweise (leider immer noch) eine DOS-Zeichencodierung (in der Regel Codepage 437 oder 850). Diese (veralteten) Codierungen kennen nur die 8-Bit-Standard-Zeichen. Der (16-Bit-)Unicode-Wert des Euro-Zeichens ist aber größer als 255 und kann folglich mit der normalen Codierung nicht dargestellt werden. Die Lösung des Problems ist zum einen die explizite Angabe der Codierung über `Console.OutputEncoding = Encoding.UTF8`. Zusätzlich dazu müssen Sie die für das Konsolenfenster verwendete Schriftart auf eine Schrift ändern, die das Euro-Zeichen kennt. Stellen Sie die Schriftart in den Eigenschaften des Konsolenfensters (Rechtsklick auf dem Titel) auf »Lucida Console«, wählen Sie eine größere Schriftgröße (14 bis 16 ist ok) und übernehmen Sie die Änderung idealerweise für alle Konsolenfenster mit demselben Titel.

8 Zahlen kaufmännisch runden

Die `Round`-Methode der `Math`-Klasse erlaubt das mathematische und das kaufmännische Runden von Zahlen. Wenn Sie neben der zu rundenden Zahl und den Dezimalstellen, auf die gerundet werden soll, nichts weiter angeben, wird mathematisch gerundet:

```
double result1 = Math.Round(1.15, 1); // 1.2
double result2 = Math.Round(1.25, 1); // 1.2
double result3 = Math.Round(1.251, 1); // 1.3
```

Listing 8: Mathematisches Runden von Zahlen

Das mathematische Rundungsverfahren stellt sicher, dass keine Ungleichheit beim Runden entsteht. Beim kaufmännischen Rundungsverfahren wird immer aufgerundet, wenn die Ziffer rechts von der Stelle, auf die gerundet wird, eine Fünf ist. Befindet sich rechts neben der Stelle, auf die gerundet wird, aber lediglich eine 5 und keine weiteren Ziffern, liegt die Zahl genau zwischen der nächsthöheren und der nächstniedrigeren gerundeten Zahl. 1,25 liegt z. B. genau zwischen 1,2 und 1,3, wenn auf eine Stelle hinter dem Komma gerundet wird. Da in solchen Fällen immer aufgerundet wird, werden mehr Zahlen auf- als abgerundet (was den Kaufleuten nur recht sein kann ☺).

Das mathematische Rundungsverfahren löst dieses Problem, indem aufgerundet wird, wenn die Ziffer an der Stelle, auf die gerundet werden soll, ungerade ist. Ist die Ziffer gerade, wird abgerundet. Das gilt natürlich nur dann, wenn hinter der Stelle, auf die gerundet werden soll, lediglich die Ziffer 5 folgt. 1,15 wird demnach auf 1,2 aufgerundet, 1,25 wird auf 1,2 abgerundet. Im Beispiel resultiert deshalb für alle Zahlen das Ergebnis 1,2. 1.251 würde allerdings auf 1,3 aufgerundet werden, da hinter der 5 noch eine Ziffer folgt.

Wenn Sie kaufmännisch runden wollen bzw. müssen, können Sie am letzten Argument der `Round`-Methode den Wert `AwayFromZero` der Aufzählung `MidpointRounding` angeben:

```
double result1 = Math.Round(1.15, 1, MidpointRounding.AwayFromZero);
// 1.2
double result2 = Math.Round(1.25, 1, MidpointRounding.AwayFromZero);
// 1.3
double result3 = Math.Round(1.251, 1, MidpointRounding.AwayFromZero);
// 1.3
```

Listing 9: Kaufmännisches Runden von Zahlen

Abbildung 15 zeigt die Ausgabe eines Testprogramms.



Abbildung 15: Mathematisches und kaufmännisches Runden über Math.Round

9 Den kleinsten und den größten Wert eines numerischen Datentyps ermitteln

Den kleinsten Wert eines numerischen Datentyps erhalten Sie über die statische `MinValue`-Eigenschaft, die alle numerischen Typen und `DateTime` besitzen. Den größten Wert können Sie über die `MaxValue`-Eigenschaft ermitteln:

```
int minValue = int.MinValue;
int maxValue = int.MaxValue;
```

10 Gradmaß in das Bogenmaß umrechnen

Die trigonometrischen Methoden der `Math`-Klasse, denen ein Winkel übergeben wird (`Sin`, `Cos`, `Tan` etc.), erwarten diesen Winkel im Bogenmaß (Radiant, Abkürzung: rad). Falls Sie (wie es im Allgemeinen üblich ist) mit dem Gradmaß (0 bis 360°) arbeiten, müssen Sie dieses in das Bogenmaß umrechnen. Im Bogenmaß entspricht der volle Winkel eines Kreises (360°) 2 * π. Daraus ergibt sich die folgende Umrechnung:

```
public static double ConvertDegreesToRadians(this double degrees)
{
    return (Math.PI / 180) * degrees;
}
```

Listing 10: Methode zur Umrechnung eines Winkels in das Bogenmaß

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

11 Bogenmaß in das Gradmaß umrechnen

Die trigonometrischen Methoden der `Math`-Klasse, die einen Sinus-, Cosinus-, Tangens- etc. Wert in einen Winkel umrechnen (z. B. `Asin`, `Acos`, `Atan`), geben den berechneten Winkel im Bogenmaß (siehe Rezept 10) zurück. Sie wollen aber wahrscheinlich eher mit dem Gradmaß

arbeiten. Die folgende Methode, die den Import des Namensraums `System` erfordert, rechnet eine Angabe im Bogenmaß in das Gradmaß um:

```
public static double ConvertRadiansToDegrees(this double radians)
{
   return (180 / Math.PI) * radians;
}
```

Listing 11: Methode zur Umrechnung einer Angabe im Bogenmaß in das Gradmaß

12 Umstellen der aktuellen Kultur

Wenn Sie Zahlen oder Datumswerte in Strings konvertieren, werden diese entsprechend der aktuell eingestellten Kultur formatiert, sofern Sie bei der Konvertierung keine spezielle Kultur angeben. Dasselbe gilt für das Konvertieren von Strings in Zahlen oder Datumswerte, wo natürlich auch die aktuelle Kultur verwendet wird.

Die aktuelle Kultur ist definiert über ein `CultureInfo`-Objekt (aus dem Namensraum `System.Globalization`), das dem aktuellen Thread zugeordnet ist. Den aktuellen Thread erreichen Sie über die Eigenschaft `CurrentThread` der `Thread`-Klasse (aus dem Namensraum `System.Threading`), das zugeordnete `CultureInfo`-Objekt über dessen Eigenschaft `CurrentCulture`.

Die aktuelle Kultur wird beim Start einer Anwendung über die im System eingestellte Kultur ermittelt. Sie können die aktuelle Kultur jedoch auch auf eine andere setzen. Dazu müssen Sie der `CurrentThread`-Eigenschaft ein `CultureInfo`-Objekt für die gewünschte Kultur übergeben. Ein solches Objekt können Sie über die statische `CreateSpecificCulture`-Methode der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) erzeugen. Dieser Methode übergeben Sie eine der international genormten Kultur-Kurzbezeichnungen. Tabelle 2 stellt einige wichtige dieser Kurzbezeichnungen dar. Die gesamte Liste finden Sie in der Dokumentation der `CultureInfo`-Klasse.

Kurzbezeichnung	Kultur
de	Deutsch allgemein
de-DE	Deutsch in Deutschland
de-AT	Österreichisches Deutsch
de-CH	Schweizer Deutsch
fr	Französisch allgemein
it	Italienisch allgemein
es	Spanisch allgemein
en	Englisch allgemein
en-US	US-amerikanisches Englisch
en-GB	Englisch in Großbritannien
en-AU	Englisch in Australien

Tabelle 2: Einige wichtige Kultur-Kurzbezeichnungen

Nun können Sie idealerweise beim Start der Anwendung die Kultur z. B. auf die australische Variante umstellen:

```
// CultureInfo-Objekt für die australische Kultur erzeugen
// und dem aktuellen Thread zuweisen
Thread.CurrentThread.CurrentCulture =
   CultureInfo.CreateSpecificCulture("en-AU");

// Als Beispiel einige Zahlen und Datumswerte formatiert ausgeben
double number = 1234.5678;
Console.WriteLine("Zahl im australischen Format: {0:#,#0.00}", number);

DateTime now = DateTime.Now;
Console.WriteLine("Datum im australischen Format: {0}", now.ToString());
```

Listing 12: Umstellen der aktuellen Kultur

Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading` und `System.Globalization`.

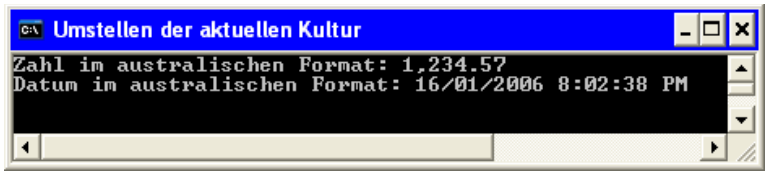


Abbildung 16: Formatierung einer Zahl und eines Datumswerts im australischen Format

13 Die verfügbaren Kulturen auslesen

Die im System verfügbaren Kulturen erhalten Sie über die statische `GetCultures`-Methode der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) in Form eines `CultureInfo`-Arrays. Diese Methode erwartet einen oder mehrere Werte der `CultureTypes`-Aufzählung, die festlegt, welche Kulturen zurückgegeben werden sollen. Der Wert `FrameworkCultures` steht z. B. für alle neutralen unspezifischen Kulturen, die im .NET Framework verfügbar sind. `NeutralCultures` steht für alle neutralen Kulturen (z. B. de), `SpecificCultures` steht für alle spezifischen (z. B. de-DE). Daneben können Sie noch die in Windows installierten Kulturen abfragen, `WindowsOnlyCultures` liefert zum Beispiel alle Kulturen, die nur in Windows zur Verfügung stehen und nicht vom .NET Framework unterstützt werden. Diese Kulturen werden Sie wohl normalerweise nicht berücksichtigen.

Ein kleiner Nachteil des zurückgehaltenen `CultureInfo`-Arrays ist, dass dieses nicht sortiert ist. Dieses Problem können Sie natürlich lösen. Ich habe dazu in Listing 13 die Klasse `CultureComparer` implementiert, die die `IComparer`-Schnittstelle implementiert. Über eine Instanz dieser Klasse wird das in den Methoden `GetAllCultures`, `GetSpecificCultures` und `GetNeutralCultures` ermittelte `CultureInfo`-Array nach dem englischen Namen der Kultur sortiert. Dieser Name ist übrigens auch der, der von der `ToString`-Methode eines `CultureInfo`-Objekts zurückgegeben wird. Wenn Sie das zurückgehaltene Array also z. B. an eine `ListBox` binden, zeigt diese automatisch den englischen Namen der Kultur an.

Zum Kompilieren der Klasse und der Methoden in Listing 13 müssen Sie die Namensräume `System`, `System.Collections.Generic` und `System.Globalization` importieren.

```
/* Vergleichsklasse für Kulturen */
internal class CultureComparer : IComparer<CultureInfo>
{
   /* Vergleicht die Namen zweier Kulturen */
   public int Compare(CultureInfo x, CultureInfo y)
   {
      return x.EnglishName.CompareTo(y.EnglishName);
   }
}

/* Gibt die verfügbaren neutralen und spezifischen Kulturen zurück */
public static CultureInfo[] GetAllCultures()
{
   // Die verfügbaren Kulturen einlesen
   System.Globalization.CultureInfo[] cultures =
      System.Globalization.CultureInfo.GetCultures(
      System.Globalization.CultureTypes.FrameworkCultures);

   // Sortieren
   Array.Sort(cultures, new CultureComparer());

   // Ergebnis zurückgeben
   return cultures;
}

/* Gibt die verfügbaren speziellen Kulturen zurück */
public static CultureInfo[] GetSpecificCultures()
{
   // Die verfügbaren Kulturen einlesen
   System.Globalization.CultureInfo[] cultures =
      System.Globalization.CultureInfo.GetCultures(
      System.Globalization.CultureTypes.SpecificCultures);

   // Sortieren
   Array.Sort(cultures, new CultureComparer());

   // Ergebnis zurückgeben
   return cultures;
}
```

Listing 13: Methoden zum sortierten Auslesen der verfügbaren Kulturen

```
/* Gibt die verfügbaren neutralen Kulturen zurück */
public static CultureInfo[] GetNeutralCultures()
{
   // Die verfügbaren Kulturen einlesen
   System.Globalization.CultureInfo[] cultures =
      System.Globalization.CultureInfo.GetCultures(
      System.Globalization.CultureTypes.NeutralCultures);

   // Sortieren
   Array.Sort(cultures, new CultureComparer());

   // Ergebnis zurückgeben
   return cultures;
}
```

Listing 13: Methoden zum sortierten Auslesen der verfügbaren Kulturen (Forts.)

Abbildung 17 zeigt eine Beispielanwendung, bei der alle speziellen Kulturen in eine `ListBox` gelesen wurden. Auf der rechten Seite wird ein Datum über einen Timer mit der aktuellen Kultur formatiert ausgegeben.

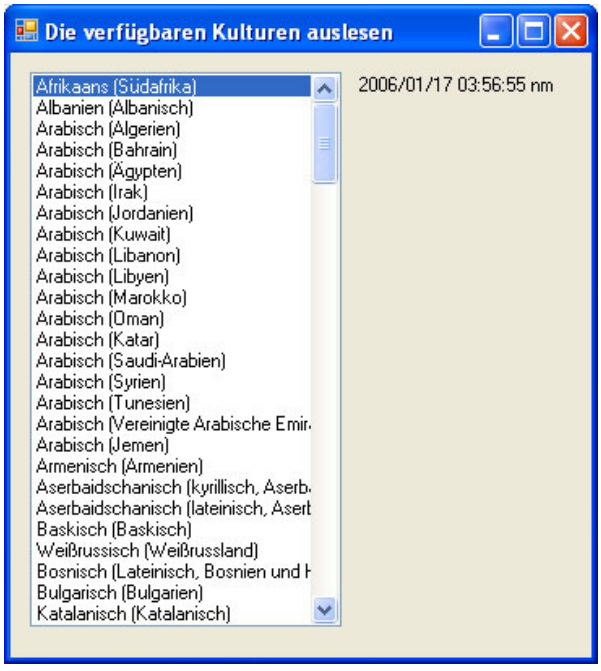


Abbildung 17: Alle speziellen Kulturen in einer ListBox

14 Zahlen in verschiedenen Länderformaten darstellen

In einigen Fällen sollen Zahlen nur für spezielle Ausgaben in länderspezifischen Formaten ausgegeben werden. Die Umstellung der aktuellen Kultur, die ich im Rezept 12 beschreibe, ist dazu nicht sinnvoll geeignet.

Den Methoden zur Formatierung können Sie aber an einem Argument ein Objekt übergeben, das die Schnittstelle `IFormatProvider` implementiert. Da die Klasse `CultureInfo` diese Schnittstelle implementiert, können Sie an diesem Argument eine Instanz dieser Klasse übergeben um Zahlen in einem bestimmten Länderformat auszugeben. Informationen zur Erzeugung eines solchen Objekts finden Sie im Rezept 12.

Das Programm in Listing 14 zeigt, wie Sie eine Zahl über deren `ToString`-Methode im australischen Format ausgeben. Das Programm benötigt den Import der Namensräume `System` und `System.Globalization`.

```
double number = 1234.5678;

// CultureInfo-Objekt für die australische Kultur erzeugen
CultureInfo aussieCulture = CultureInfo.CreateSpecificCulture("en-AU");

// Zahl in das australische Format formatieren
string formattedNumber = number.ToString("#,#0.00", aussieCulture);
```

Listing 14: Explizites Formatieren einer Zahl im australischen Format

15 Einen Bytewert in ein besser lesbares Format umwandeln

Wenn Sie in einem Programm Werte ermitteln, deren Einheit Byte ist, und Sie diese im Programm ausgeben wollen, fördert es die Lesbarkeit, wenn Sie große Werte in deren KB-, MB- oder GB-Entsprechung umwandeln. Dazu müssen Sie lediglich ermitteln, ob der Bytewert jeweils kleiner ist als ein KB, MB bzw. GB, den Wert dann entsprechend teilen und das Kürzel für die Einheit anhängen.

Die Methode `ToReadableFormat` in Listing 15 übernimmt diese Aufgabe. Um eine möglichst hohe Flexibilität anzubieten, erwartet die Methode am zweiten Argument einen Formatstring, über den die übergebene Zahl nach der Umrechnung formatiert wird.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

```
public static string ToReadableFormat(this long byteValue, string format)
{
   if (byteValue < 1024)
   {
      return byteValue + " Byte";
```

Listing 15: Methode zum Formatieren eines Bytewerts in ein besser lesbares Format

```
    }
    if (byteValue < 1048576) // 1048576 = 1024 * 1024
    {
        return String.Format("{0:" + format + "} KB", byteValue / 1024F);
    }
    if (byteValue < 1073741824) // 1073741824 = 1024 * 1024 * 1024
    {
        return String.Format("{0:" + format + "} MB", byteValue / 1048576F);
    }
    else
    {
        return String.Format("{0:" + format + "} GB", byteValue / 1073741824F);
    }
}
```

Listing 15: Methode zum Formatieren eines Bytewerts in ein besser lesbares Format (Forts.)

`ToReadableFormat` gibt den u. U. umgerechneten Wert mit der hinten angehängten Einheit als Zeichenkette zurück. Aus dem Wert 1023 resultiert z. B. der String "1023 Byte", aus 2222 wird "2,1699 KB" (bei einer Formatierung mit vier Dezimalstellen). Der folgende Programmcode zeigt Beispiele für eine Anwendung dieser Methode:

```
long number = 1023;
Console.WriteLine(number.ToReadableFormat("#,#0.####"));
number = 1024;
Console.WriteLine(number.ToReadableFormat("#,#0.00"));
number = 2222;
Console.WriteLine(number.ToReadableFormat("#,#0.####"));
number = 3333 * 1024;
Console.WriteLine(number.ToReadableFormat("#,#0.####"));
number = 4444L * 1024 * 1024;
Console.WriteLine(number.ToReadableFormat("#,#0.####"));
```

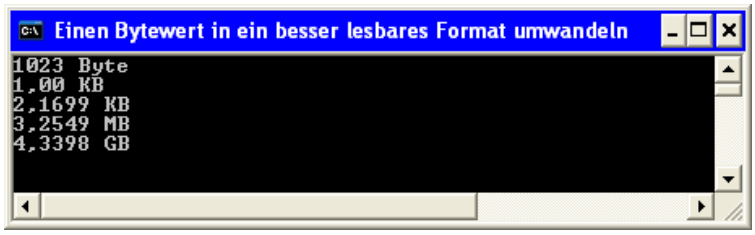


Abbildung 18: Die Ausgabe des Beispielprogramms

16 Überprüfen, ob ein String eine gültige Zahl speichert

Immer dann, wenn ein Anwender in einem Programm Zahlen eingeben soll und diese für Berechnungen weiterverarbeitet werden, sollten Sie die eingegebenen Zeichenketten daraufhin überprüfen, ob diese eine gültige Zahl speichern. Die Konvertierung eines Strings, der keine gültige Zahl verwaltet (über die Methoden der `Convert`-Klasse oder die `Parse`-Methoden der

numerischen Datentypen), in einen numerischen Datentyp führt ansonsten zu einer Ausnahme vom Typ `FormatException`.

Die einfachste Variante, einen String daraufhin zu überprüfen, ob er eine gültige Zahl speichert, ist, diesen einfach zu konvertieren und die Ausnahme, die beim Konvertieren ungültiger Strings erzeugt wird, abzufangen:

```
Console.Write("Geben Sie eine Zahl ein: ");
string input = Console.ReadLine();
int number = 0;
try
{
   number= Convert.ToInt32(input);
   Console.WriteLine("Die Eingabe ist gültig: " + number);
}
catch (FormatException)
{
   Console.WriteLine("Die Eingabe ist ungültig");
}
```

Listing 16: Überprüfen einer Eingabe über das Abfangen der bei ungültigen Werten geworfenen FormatException

Eine andere Möglichkeit ist, die `TryParse`-Methode zu verwenden, die die numerischen und Datums-Typen besitzen. `TryParse` versucht eine Konvertierung des übergebenen Werts und gibt bei Erfolg `true` zurück. In diesem Fall wird der konvertierte Wert in die am zweiten `out`-Argument übergebene Variable geschrieben:

```
Console.Write("Geben Sie eine Datum ein: ");
string input = Console.ReadLine();
DateTimeOffset date;
if (DateTimeOffset.TryParse(input, out date))
{
    Console.WriteLine("Die Eingabe ist gültig: " + date);
}
else
{
    Console.WriteLine("Die Eingabe ist ungültig")
}
```

Listing 17: Überprüfen einer Eingabe über die TryParse-Methode

Ein kleiner Nachteil dieser Methode ist, dass `TryParse` etwas langsamer ausgeführt wird als die Prüfung der `Exception` (vergleichen Sie hierzu das Beispielprogramm zu diesem Rezept). Das Ganze bewegt sich aber im Mikrosekundenbereich, weswegen Sie diesen Nachteil wohl vernachlässigen können. Ein Vorteil der `TryParse`-Methode ist, dass Sie optional ein Objekt übergeben können, das die `IFormatProvider`-Schnittstelle implementiert, um zu erreichen, dass

beim Parsen eine bestimmte Formatierung verwendet wird. Am einfachsten ist die Übergabe eines `CultureInfo`-Objekts zur Berücksichtigung der dadurch bestimmten Kultur. Da dies nur bei Zahlen mit Dezimalstellen Sinn macht, verwende ich für das folgende Beispiel eine `double`-Zahl.

Der `TryParse`-Methode übergeben Sie wieder am ersten Argument den zu überprüfenden String. Am zweiten Argument können Sie in der zweiten Variante dieser Methode mit Werten der Aufzählung `System.Globalization.NumberStyles` festlegen, welches Format die eingegebene Zahl besitzen darf. Dieses Argument ist ein Bitfeld, Sie können also mehrere der `NumberStyles`-Konstanten über | kombinieren. Über die Konstante `AllowDecimalPoint` legen Sie z. B. fest, dass ein Dezimaltrennzeichen erlaubt ist, `AllowThousands` erlaubt ein Tausender-Trennzeichen, `AllowLeadingSign` ein führendes Vorzeichen und `AllowCurrencySymbol` ein Währungssymbol. Mit diesen und den anderen Konstanten können Sie recht genau festlegen, welches Format die Zahl besitzen darf. `NumberStyles` besitzt daneben aber auch Konstanten für vordefinierte Formate, die sich größtenteils über ihren Namen erklären: `Currency`, `Float`, `HexNumber`, `Integer` und `Number` (wie `Float`, nur mit Tausender-Trennzeichen).

Am dritten Argument übergeben Sie der zweiten Variante der `TryParse`-Methode ein Objekt, das die `IFormatProvider`-Schnittstelle implementiert. Das kann z. B. ein `CultureInfo`-Objekt sein. Für die aktuelle Kultur übergeben Sie `System.Threading.Thread.CurrentThread.CurrentCulture`.

Am letzten Argument (das mit `out` deklariert ist) übergeben Sie schließlich eine `double`-Variable, in die `TryParse` den konvertierten Wert schreibt.

Das folgende Beispiel erwartet die Eingabe einer `double`-Zahl in der allgemeinen englischen Schreibweise:

```
double doubleResult;
if (double.TryParse(input, NumberStyles.Float,
    CultureInfo.CreateSpecificCulture("en"), out doubleResult))
{
    Console.WriteLine("Die Eingabe ist gültig");
}
else
{
    Console.WriteLine("Die Eingabe ist ungültig")
}
```

Eine dritte Möglichkeit, eine Eingabe auf Gültigkeit zu überprüfen, wäre übrigens die Verwendung der `IsNumeric`-Methode von Visual Basic, aber dazu müssten Sie die Assembly *Microsoft.VisualBasic* referenzieren (was wir als C#-Entwickler ja nur sehr ungern machen):

```
if (Microsoft.VisualBasic.Information.IsNumeric(input))
{
    Console.WriteLine("Die Eingabe ist gültig");
}
else
{
    Console.WriteLine("Die Eingabe ist ungültig")
}
```

`IsNumeric` überprüft lediglich auf `double`-Zahlen und ermöglicht deshalb nicht, zu überprüfen, ob die Eingabe eine Ganzzahl ist.

In einigen Fällen, wenn Sie Eingaben lediglich prüfen und nicht gleich auch in eine Variable schreiben wollen, ist `TryParse` u. U. zu umständlich. Soll eine spezielle Kultur berücksichtigt werden, ist `TryParse` ebenso umständlich, da Sie (in meinen Augen unsinnigerweise) neben dem `IFormatProvider`-Objekt das Format der Zahl mit übergeben müssen. Außerdem kann es bei der Verwendung ohne die Übergabe eines `NumberStyles`-Werts zu einem Problem kommen:

Da die `TryParse`-Methode grundsätzlich alle zum Datentyp passenden Formate erlaubt (bei `double` z. B. neben dem normalen Format auch die Exponentialschreibweise), ist die Überprüfung einerseits sehr flexibel. Andererseits gibt `TryParse` in der deutschen Kultur z. B. auch `true` zurück, wenn die Zahl falsch gesetzte Tausender-Trennzeichen enthält. Der String "1.2" ist z. B. gültig (und ergibt bei der Konvertierung die Zahl 12). Da Anwender in der Praxis häufig versehentlich oder sogar bewusst den Punkt als Dezimaltrennzeichen verwenden, ist die Überprüfung mit `TryParse` ohne weitere Vorkehrungen in vielen Fällen gefährlich. Besser wäre es, Tausender-Trennzeichen von vornherein auszuschließen. Und das können Sie nur, indem Sie `NumberStyles.Float` am zweiten Argument übergeben, was dann aber neben der Rückgabe-Variable am vierten auch die Übergabe des dritten Arguments erzwingt.

Also habe ich das Ganze ein wenig vereinfacht und einige Methoden geschrieben, die eine Eingabe einfach daraufhin prüfen, ob diese gültig ist. Diese Methoden verwenden für Ganzzahlwerte die schnellere Konvertierung mit `Exception`-Prüfung. Für Werte mit Nachkommastellen verwenden die Methoden `TryParse`. Um Probleme mit Tausender-Trennzeichen zu verhindern, werden diese ausgeschlossen, indem `TryParse` als Zahl-Stil bei Double-Zahlen `NumberStyles.Float` übergeben wird.

Zur Verwendung dieser Methoden müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
/* Überprüft einen String daraufhin, ob dieser einen double-Wert speichert */
public static bool IsDouble(this string value)
{
   double result;
   return double.TryParse(value, NumberStyles.Float, null, out result);
}

/* Überprüft einen String daraufhin, ob dieser einen double-Wert speichert */
public static bool IsDouble(this string value, IFormatProvider provider)
{
   double result;
   return double.TryParse(value, NumberStyles.Float, provider, out result);
}

/* Überprüft einen String daraufhin, ob dieser einen int-Wert speichert */
public static bool IsInt(this string value)
```

Listing 18: Methoden zur Überprüfung eines Strings auf numerische Daten

```
{
   try
   {
      Convert.ToInt32(value);
      return true;
   }
   catch
   {
      return false;
   }
}

/* Überprüft einen String daraufhin, ob dieser einen int-Wert speichert */
public static bool IsInt(this string value, IFormatProvider provider)
{
   try
   {
      Convert.ToInt32(value, provider);
      return true;
   }
   catch
   {
      return false;
   }
}
```

Listing 18: Methoden zur Überprüfung eines Strings auf numerische Daten (Forts.)

Wenn die Methoden `true` zurückgeben, können Sie den String sicher konvertieren.

17 Überprüfen, ob ein String ein gültiges Datum speichert

Ähnlich wie bei eingegebenen Zahlen ist es sinnvoll, vom Anwender eingegebene Datumswerte vor der Weiterverarbeitung zu prüfen. Dazu können Sie eine ähnliche Technik verwenden wie bei der Überprüfung auf die Gültigkeit einer Zahl. Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
/* Überprüft, ob der übergebene String ein gültiges Datum speichert */
public static bool IsDate(this string dateString)
{
   try
   {
      Convert.ToDateTime(dateString);
      return true;
   }
```

Listing 19: Methoden zur Überprüfung der Gültigkeit eines Datums-Strings

```
   catch
   {
      return false;
   }
}

/* Überprüft, ob der übergebene String ein gültiges Datum speichert */
public static bool IsDate(this string dateString, IFormatProvider provider)
{
   try
   {
      Convert.ToDateTime(dateString, provider);
      return true;
   }
   catch
   {
      return false;
   }
}
```

Listing 19: Methoden zur Überprüfung der Gültigkeit eines Datums-Strings (Forts.)

18 Zufallszahlen erzeugen

Zufallszahlen können Sie über eine Instanz der Klasse `System.Random` erzeugen. Die Methode `Next` liefert eine zufällig ermittelte `int`-Zahl zurück. Rufen Sie `Next` ohne Argument auf, liegt die Zahl im Bereich zwischen 0 und 2.147.483.647 (dem maximalen `int`-Wert). Sie können dieser Methode aber auch einen Maximalwert oder einen Minimal- und einen Maximalwert übergeben. So können Sie z. B. Zufallszahlen im Bereich von 1 bis 42 erzeugen.

```
Random random = new Random();
int number1 = random.Next(1, 42);
int number2 = random.Next(1, 42);
int number3 = random.Next(1, 42);
```

Die erzeugten Zufallszahlen sind gleich verteilt. Wenn Sie also sehr viele Zahlen erzeugen, ist die Wahrscheinlichkeit groß, dass die einzelnen Zahlen mit einer gleichen Anzahl erzeugt werden.

Die `Random`-Klasse ist für die Generierung echter Zufallszahlen allerdings nicht geeignet: `Random` verwendet eine bestimmte Anzahl festgelegter Zahlenreihen mit `double`-Werten zwischen 0 und 1 als Basis für die Erzeugung der Zufallszahlen. Die nächste Basiszahl können Sie übrigens über die Methode `NextDouble` abrufen. Ein so genannter Seed-Wert (Seed = Saat) bestimmt dabei, welche dieser Zahlenreihen verwendet wird. Im Konstruktor der `Random`-Klasse können Sie den Seed-Wert als `int`-Zahl übergeben. Nun ist es logisch, dass zwei `Random`-Instanzen mit demselben Seed-Wert exakt dieselben Basiszahlen zurückliefern. Identische `Next`-Aufrufe ergeben dann also immer (auch auf verschiedenen Rechnern) identische Zahlenreihen. Mit einem Seed-Wert von 1 erzeugt `Next(1, 42)` z. B. bei drei Aufrufen immer die Folge 11, 5 und 21.

Der Default-Konstruktor der `Random`-Klasse verwendet deswegen die aktuellen Systemticks (`Environment.TickCount`) als Seed-Wert. Damit ist die Wahrscheinlichkeit, identische Zahlenreihen zu erzeugen, einigermaßen gering, aber nicht ausgeschlossen. Wenn Sie mit mehreren `Random`-Instanzen arbeiten und diese direkt hintereinander erzeugen, kann es auf schnellen Rechnern sogar vorkommen, dass diese dieselbe Systemzeit verwenden und damit identische Zahlenreihen liefern. In diesem Fall sollten Sie dem vorbeugen und den Seed-Wert für jede Instanz selbst berechnen (z. B. indem Sie einfach eine beliebige Zahl zu den ausgelesenen Ticks addieren).

Wenn Sie sichere Zufallszahlen erzeugen wollen, können Sie die Klasse `RNGCryptoServiceProvider` aus dem Namensraum `System.Security.Cryptography` verwenden. Diese Klasse wird normalerweise für die Implementierung von speziellen Verschlüsselungsverfahren verwendet.

Die Methode `GetBytes` eines `RNGCryptoServiceProvider`-Objekts füllt ein übergebenes `byte`-Array mit »kryptografisch starken« Zufallszahlen. `GetNonZeroBytes` arbeitet ähnlich, füllt aber nur Zahlen größer 0 in die Arrayelemente. Das `byte`-Array müssen Sie zuvor mit der Größe der Anzahl der zu erzeugenden Zahlen erzeugen. Damit können Sie Zufallszahlen im Bereich von 0 bzw. 1 und 255 erzeugen:

```
Console.WriteLine("Zufallszahlen zwischen 1 und 255");
RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
byte[] randomNumbers = new byte[10];
rng.GetNonZeroBytes(randomNumbers);
for (int i = 0; i < 10; i++)
{
   Console.WriteLine(randomNumbers[i]);
}
```

Listing 20: Erzeugen echter Zufallszahlen zwischen 1 und 255

Um größere Zahlen zu erhalten, können Sie einfach mehrere Arrays füllen und die einzelnen Elemente miteinander multiplizieren:

```
Console.WriteLine("Zufallszahlen zwischen 1 und 65025");
byte[] randomNumbers1 = new byte[10];
byte[] randomNumbers2 = new byte[10];
rng.GetNonZeroBytes(randomNumbers1);
rng.GetNonZeroBytes(randomNumbers2);
for (int i = 0; i < 10; i++)
{
   Console.WriteLine(randomNumbers1[i] * randomNumbers2[i]);
}
```

Listing 21: Erzeugen echter Zufallszahlen zwischen 1 und 65025

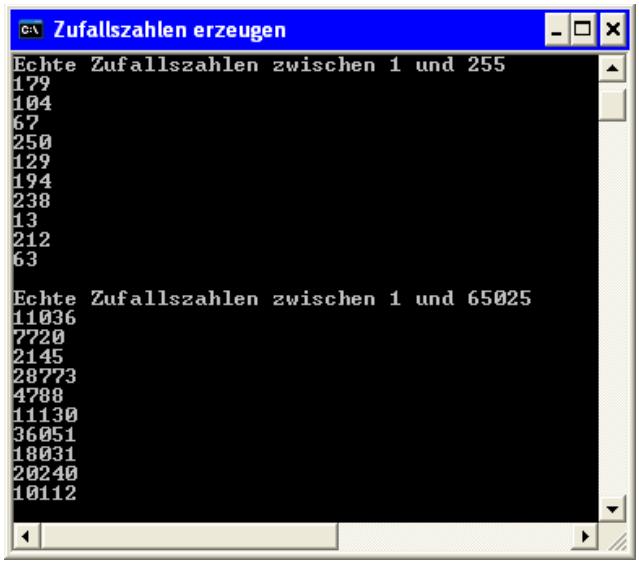


Abbildung 19: Erzeugen echter Zufallszahlen

Wenn Sie echte Zufallszahlen in einem bestimmten Bereich erhalten wollen, können Sie die ermittelte Zahl folgendermaßen umrechnen:

```
Divisor = (Zufallszahlen-Bereichsumfang / (Endzahl - Startzahl + 1))
Zahl = (byte)((Zahl / Divisor) + Startzahl)
```

Wenn Sie Zufallszahlen im Bereich von 0 bis 255 erzeugen und Zahlen im Bereich von 10 bis 20 erhalten wollen, sieht die Berechnung der einzelnen Zahlen folgendermaßen aus:

```
double divisor = 256 / 20;
zahl = (byte)((zahl / divisor ) + 10);
```

Die Methode `GetRandomByteNumber` in Listing 22 zeigt dies für Zahlen im Byte-Bereich. Diese Methode benötigt den Import der Namensräume `System` und `System.Security.Cryptography`.

```
public static byte[] GetRandomNumbers(int count, byte min, byte max)
{
   // Zufallszahlen erzeugen
   RNGCryptoServiceProvider csp = new RNGCryptoServiceProvider();
   byte[] numbers = new Byte[count];
   csp.GetBytes(numbers);

   // Die Zahlen umrechnen
   double divisor = 256F / (max - min + 1);
   if (min > 0 || max < 255)
   {
      for (int i = 0; i < count; i++)
```

Listing 22: Erzeugen echter Zufallszahlen in einem gegebenen Bereich

```
      {
         numbers[i] = (byte)((numbers[i] / divisor) + min);
      }
   }

   return numbers;
}
```

Listing 22: Erzeugen echter Zufallszahlen in einem gegebenen Bereich (Forts.)

Ein Test dieser Methode hat bei 10.000 erzeugten Zahlen eine recht gute Gleichverteilung ergeben. Sie finden diesen Test als Teil des Beispiels zu diesem Rezept auf der Buch-CD.

19 High- und Low-Word lesen

Einige Windows-API-Funktionen geben numerische Daten in einem (normalerweise vorzeichenlosen) 32-Bit-Integerwert zurück, dessen hohes und niedriges Wort (ein Wort hat 16 Bit) unterschiedliche Informationen speichern. Der `uint`-Wert `0xAAAAFFFF` speichert z. B. im High-Word den Wert `0xAAAA` und im Low-Word den Wert `0xFFFF`.

Sie können das hohe Wort eines vorzeichenlosen 32-Bit-Werts extrahieren, indem Sie den Wert um 16 Bit nach rechts shiften. Das niedrige Wort erhalten Sie über eine bitweise Addition mit `0xFFFF`.

Die Methoden `GetHighWord` und `GetLowWord` in Listing 23 gehen auf diese Weise vor um das hohe und das niedrige Wort in je einer Variante aus einem `uint`-Wert bzw. `int`-Wert zu extrahieren. Das Ergebnis der Berechnung wird bei der Rückgabe in einen `ushort`-Wert umgewandelt, da die Berechnung selbst einen `uint`- bzw. `int`-Wert ergibt.

`GetHighWord` und `GetLowWord` erfordern den Import des Namensraums `System`.

```
/* Extrahiert das High-Word aus einem uint-Wert */
public static ushort GetHighWord(this uint value)
{
   return (ushort)(value >> 16);
}

/* Extrahiert das Low-Word aus einem uint-Wert */
public static ushort GetLowWord(this uint value)
{
   return (ushort)(value & 0xFFFF);
}

/* Extrahiert das High-Word aus einem int-Wert */
public static ushort GetHighWord(this int value)
```

Listing 23: Methoden zum Extrahieren des hohen und des niedrigen Worts eines 32-Bit-Integerwerts

```
{
   return (ushort)(value >> 16);
}

/* Extrahiert das Low-Word aus einem int-Wert */
public static ushort GetLowWord(this int value)
{
   return (ushort)(value & 0xFFFF);
}
```

Listing 23: Methoden zum Extrahieren des hohen und des niedrigen Worts eines 32-Bit-Integerwerts (Forts.)

Abbildung 20 zeigt die Ausgabe eines Programms, das das High- und das Low-Word über die Methoden in Listing 23 für die `uint`-Zahl `0xAAAAFFFF` berechnet. Diese Zahl wurde dann über einen Typecast in eine `int`-Variable geschrieben und damit ebenfalls das High- und das Low-Word berechnet.

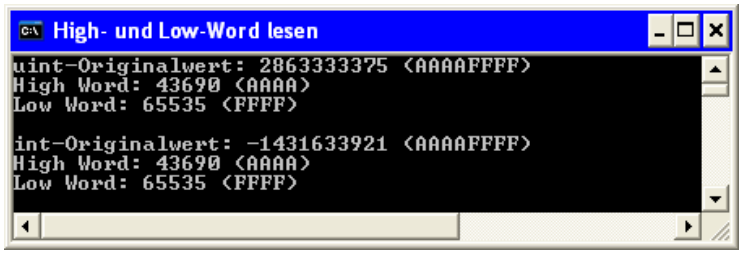


Abbildung 20: Berechnung des High- und des Low-Word einer uint- und einer int-Zahl

20 High- und Low-Word setzen

Wenn Sie Windows-API-Funktionen aufrufen, die einen 32-Bit-Integerwert erwarten, dessen Low- und High-Word eine separate Bedeutung besitzen, müssen Sie diese separat setzen. Dazu können Sie zwei `ushort`-Variablen verwenden, die die Werte des hohen und des niedrigen Worts speichern. Zum Setzen des hohen Worts shiften Sie die `ushort`-Variable um 16 Bit nach links und addieren das Ergebnis mit dem um das hohe Wort reduzierten Integerwert. Das hohe Wort des Integerwerts erhalten Sie über eine bitweise Addition mit `0xFFFF`. Damit erreichen Sie, dass nur das hohe Wort gesetzt und der Wert des niedrigen Worts nicht geändert wird. Die Methode `SetHighWord` in Listing 24, die in je einer `uint`- und einer `int`-Variante implementiert ist, zeigt, wie diese Berechnung ausgeführt wird. Das Ergebnis des Linksshift muss dabei in einen `uint`- bzw. `int`-Wert umgewandelt werden, da dieses den Typ `long` besitzt.

Um das niedrige Wort zu setzen, addieren Sie den Integerwert bitweise mit `0xFFFF0000`, um den Wert dieses Worts zu löschen, und addieren einfach das neue Wort auf das Ergebnis. Die Methode `SetLowWord` in Listing 24 zeigt, wie das ausgeführt wird. In der `int`-Variante dieser Methode ergibt die bitweise Addition einen `long`-Wert. Das liegt daran, dass der Wert `0xFFFF0000` vom Compiler nicht als `uint`, sondern als `long` ausgewertet wird, weil ein `int`-Wert in der Berechnung enthalten ist. Deshalb muss das Ergebnis der bitweisen Addition in einen `int`-Wert umgewandelt werden. In der `uint`-Variante ist diese Umwandlung nicht notwendig,

da der Compiler den Wert `0xFFFF0000` hier als `uint` auswertet, weil die Addition ebenfalls mit einem `uint`-Wert erfolgt.

Zum Kompilieren dieser Methoden müssen Sie den Namensraum `System` importieren.

```
/* Setzt das High-Word in einer uint-Variablen */
public static uint SetHighWord(this uint value, ushort highWord)
{
   return (value & 0xFFFF) + (uint)(highWord << 16);
}

/* Setzt das Low-Word in einer uint-Variablen */
public static uint SetLowWord(this uint value, ushort lowWord)
{
   return (value & 0xFFFF0000) + lowWord;
}

/* Setzt das High-Word in einer int-Variablen */
public static int SetHighWord(this int value, ushort highWord)
{
   return (value & 0xFFFF) + (int)(highWord << 16);
}

/* Setzt das Low-Word in einer uint-Variablen */
public static int SetLowWord(this int value, ushort lowWord)
{
   return (int)(value & 0xFFFF0000) + lowWord;
}
```

Listing 24: Methoden zum Setzen des High- und des Low-Word in einem uint- oder int-Wert

21 Beep ausgeben

Piepstöne können Sie über die Klasse `System.Media.SystemSounds` in unterschiedlicher Form ausgeben:

```
System.Media.SystemSounds.Beep.Play(); // Einfacher Beep
System.Media.SystemSounds.Question.Play(); // Frage-Piepston
System.Media.SystemSounds.Asterisk.Play(); // Warnungs-Piepston 1
System.Media.SystemSounds.Exclamation.Play(); // Warnungs-Piepston 2
System.Media.SystemSounds.Hand.Play(); // Fehler-Piepston
```

Listing 25: Ausgabe der vier verschiedenen System-Piepstöne

Das Windows-API bietet übrigens über die Funktion `Beep` noch eine weitere Möglichkeit, einen Meldungston auszugeben. Dieser Funktion können Sie eine Frequenz und einen Wert übergeben, der die Länge des auszugebenden Tons in Millisekunden angibt. Leider funktioniert

`Beep` nur dann, wenn keine Soundkarte installiert ist, da diese Funktion den Lautsprecher des Computers anspricht.

Hinweis

In den Rezepten 313, 314 und 315 finden Sie Methoden bzw. Klassen zum Abspielen von Wave- und anderen Multimedia-Dateien. Darüber können Sie dem Anwender auch akustisch mitteilen, dass ein besonderer Zustand eingetreten ist.

22 Strings ohne Berücksichtigung der Groß- bzw. Kleinschreibung vergleichen

Beim direkten Vergleich zweier Strings berücksichtigt der Vergleichsoperator der `string`-Klasse die Schreibweise der einzelnen Zeichen. Der String »Hallo« ist bei einem solchen Vergleich ungleich dem String »hallo«. Wenn Sie zwei Strings ohne Berücksichtigung der Groß-/Kleinschreibung vergleichen wollen, können Sie diese einfach mit der `CompareTo`-Methode der `String`-Klasse vergleichen: Geben Sie neben den zu vergleichenden Strings am dritten Argument (`ignoreCase`) `true` an. `Compare` gibt 0 zurück, wenn beide Strings gleich sind.

```
string string1 = "Hallo";
string string2 = "hallo";

if (String.Compare(string1, string2, true) == 0)
{
   Console.WriteLine("'{0}' und '{1}' sind gleich", string1, string2);
}
else
{
   Console.WriteLine("'{0}' und '{1}' sind nicht gleich", string1, string2);
}
```

Listing 26: Case-insensitiver Vergleich von zwei Zeichenketten

Hinweis

Der ebenfalls denkbare Vergleich über ein Umwandeln der Strings in Klein- oder Großschreibung (über `ToLower` bzw. `ToUpper`) ist übrigens nicht zu empfehlen, da `Compare` wesentlich effizienter arbeitet. Ein Vergleich mit `Compare` ist nur unwesentlich langsamer als ein direkter Vergleich.

23 Teilstrings flexibel ersetzen

Wenn Sie in einem String Teilstrings ersetzen wollen, bieten sich Ihnen gleich mehrere Möglichkeiten, die mehr oder weniger einfach anzuwenden oder flexibel sind. Leider finden Sie im .NET Framework keine universelle Methode. Deshalb habe ich eine eigene `Replace`-Methode entwickelt, die sehr flexibel und zudem noch recht schnell ist. Diese Methode stelle ich im folgenden Abschnitt vor. Zuvor folgt eine Übersicht der einzelnen Ersetzungsmöglichkeiten (Tabelle 3).

Methode	Möglichkeit, die Groß-/Kleinschreibung zu ignorieren	Ersetzen in einem anzugebenden Teilstring	Möglichkeit, anzugeben, ab welchem gefundenen Suchstring ersetzt wird	Möglichkeit, anzugeben, wie viele gefundene Suchstrings ersetzt werden
`String.Replace`	Nein	Nein	Nein	Nein
`String-Builder.Replace`	Nein	Ja	Nein	Nein
`Regex.Replace`	Ja	Teilweise (die Startposition kann angegeben werden)	Nein	Ja
Visual Basic Replace	Ja	Teilweise (die Startposition kann angegeben werden)	Nein	Ja
Eigene Replace-Methode	Ja	Nein	Ja	Ja

Tabelle 3: Die Möglichkeiten, Teilstrings zu ersetzen

Was mich bei den verfügbaren `Replace`-Methoden sehr ärgert, ist die Tatsache, dass diese nicht konsistent zueinander sind. Außerdem besitzen die Argumente zur Angabe der Anzahl eine unterschiedliche Bedeutung. Einmal geben Sie damit die Anzahl der Zeichen an, die ab der Startposition berücksichtigt werden sollen (`StringBuilder.Replace`), bei anderen Methoden (`Regex.Replace`, Visual-Basic-Replace) geben Sie die Anzahl der zu ersetzenden Teilstrings an. Warum hat Microsoft die `Replace`-Methoden der einzelnen Klassen nicht einfach konsistent (und mit verschiedenen Varianten, die alle Anforderungen abdecken, überladen) entwickelt? Na ja, die hier vorgestellte `Replace`-Methode füllt diese Lücke ☺.

Die eigene Replace-Methode

Die von mir entwickelte `Replace`-Methode ermöglicht eine Ersetzung ohne Berücksichtigung der Groß-/Kleinschreibung, das Ersetzen ab einem anzugebenden gefundenen String und das Ersetzen einer bestimmten Anzahl an gefundenen Suchstrings.

Diese Methode ersetzt im übergebenen String `source` die an `find` übergebenen Teilstrings durch `replacement`. Am Argument `start` übergeben Sie, ab welcher Fundstelle ersetzt werden soll. Dieses Argument ist, wie in C# üblich, nullbasiert (0 steht für den Index der ersten Fundstelle). Wenn Sie hier z. B. 1 übergeben, wird die erste Fundstelle übersprungen und erst ab der zweiten ersetzt. Das Argument `count` gibt an, wie viele Ersetzungen vorgenommen werden sollen. -1 steht hier für alle gefundenen Teilstrings (ab `start`).

```
public static string Replace(this string source, string find,
   string replacement, bool ignoreCase, int start, int count)
{
   int pos1 = 0, pos2 = 0;
```

Listing 27: Methode zum flexiblen Ersetzen von Strings

```
int findStringLen = find.Length;
string baseString;
string result = null;

// Den Sonderfall abhandeln, dass count mit 0 angegeben ist
if (count == 0)
{
   return source;
}

// Wenn der Vergleich ohne Berücksichtigung der Groß-/Kleinschreibung
// erfolgen soll, werden der zu durchsuchende und der Suchstring einfach in
// Kleinschreibung umgewandelt
if (ignoreCase)
{
   baseString = source.ToLower();
   find = find.ToLower();
}
else
{
   baseString = source;
}

// Erstes Vorkommen des Suchstrings suchen
pos2 = baseString.IndexOf(find);

// Den String durchgehen, solange der Suchstring noch gefunden wurde
int findIndex = -1;
int replaceCount = 0;
while (pos2 != -1)
{
   findIndex++;
   if (findIndex >= start)
   {
      // Wenn der Index des gefundenen Teilstrings größer/gleich start ist:
      // Das Ergebnis zusammensetzen und die Such-Indizes auf die neuen
      // Positionen setzen
      result += source.Substring(pos1, pos2 - pos1) + replacement;
      pos1 = pos2 + findStringLen;
      pos2 = baseString.IndexOf(find, pos1);

      // Wenn count größer -1 definiert ist: Überprüfen, ob die Anzahl
      // erreicht ist
      replaceCount++;
      if (count > -1 && replaceCount >= count)
```

Listing 27: Methode zum flexiblen Ersetzen von Strings (Forts.)

```
            {
               break;
            }
         }
         else
         {
            // Index des gefundenen Teilstrings ist noch kleiner start:
            // Teilstring ohne zu ersetzen an das Ergebnis anhängen
            result += source.Substring(pos1, pos2 - pos1 + findStringLen);

            // Such-Positionen aktualisieren
            pos1 = pos2 + findStringLen;
            pos2 = baseString.IndexOf(find, pos1);
         }
      }

      if (pos1 < source.Length)
      {
         // Wenn nach dem letzten Suchstring noch ein Teilstring gespeichert ist,
         // diesen hinten anhängen
         result += source.Substring(pos1, source.Length - pos1);
      }

      // Ergebnis zurückgeben
      return result;
   }
```

Listing 27: Methode zum flexiblen Ersetzen von Strings (Forts.)

`Replace` benötigt den Import des Namensraums `System`.

`Replace` verwendet die Methoden `IndexOf` und `Substring` des `string`-Objekts um den gesuchten Teilstring zu suchen und durch den neuen Teilstring zu ersetzen. Um die Vernachlässigung der Groß-/Kleinschreibung zu verwirklichen schreibt `Replace` den übergebenen String in eine lokale Kopie, die in Kleinschrift umgewandelt wird, wenn `ignoreCase` `true` ist. In diesem Fall wird auch der Suchstring in Kleinschrift umgewandelt.

`Replace` sucht dann einfach in einer Schleife nach dem Suchstring. Findet es diesen und der Index des gefundenen Teilstrings ist größer/gleich `start`, hängt `Replace` den aktuellen Teilstring ab der letzten Startposition bis zur Position des Teilstrings an eine Ergebnis-Stringvariable an. Die Position für die weitere Suche wird dann für den nächsten Durchlauf auf die Position hinter dem Suchstring gesetzt.

In der Schleife werden jede Fundstelle und jeder Ersatz gezählt. Mit diesen Werten vergleicht `Replace` die Angaben, die an den Argumenten `start` und `count` übergeben wurden. Das Ersetzen wird nur dann ausgeführt, wenn der aktuelle Suchzähler größer oder gleich `start` ist. Ist `count` mit einem Wert größer -1 angegeben, wird die Schleife unterbrochen, wenn die Anzahl der Ersetzungen `count` entspricht. Unterhalb der Schleife wird dann noch der eventuell übrig gebliebene rechte Teilstring an das Ergebnis angehängt und dieses zurückgegeben.

Listing 28 zeigt eine Anwendung dieser Methode.

```
string source = "AAA-aaa-AAA-aaa";
string find = "aaa";
string replacement = "999";

// Alle Fundstellen ersetzen
string result = source.Replace(find, replacement, true);
Console.WriteLine("Alle Fundstellen ersetzt: {0}", result)

// Ab der zweiten Fundstelle alle weiteren ersetzen
result = source.Replace(find, replacement, true, 1, -1);
Console.WriteLine("Ab der zweiten Fundstelle alle weiteren ersetzt: {0}",
   result)

// Ab der zweiten Fundstelle eine ersetzen
result = source.Replace(find, replacement, true, 1, 1);
Console.WriteLine("Ab der zweiten Fundstelle eine Fundstelle ersetzt: {0}",
   result)
```

Listing 28: Beispiel zur Anwendung der Replace-Methode



Abbildung 21: Das Beispielprogramm in Aktion

String.Replace

Die `Replace`-Methode einer String-Instanz bietet eine einfache Möglichkeit, einzelne Zeichen oder Teilstrings im String zu ersetzen:

```
string source = "AAA-aaa-AAA-aaa";
string find = "aaa";
string replacement = "999";
string result = source.Replace(find, replacement);
```

Ein wesentlicher Nachteil dieser Methode ist, dass Sie nicht bestimmen können, dass die Groß-/Kleinschreibung nicht berücksichtigt werden soll. Das Ergebnis der Ersetzung im obigen Code ist z. B. "AAA-999-AAA-999". Die großgeschriebenen As werden nicht ersetzt. Ein (je nach Anwendung) eventueller anderer Nachteil ist, dass Sie nicht bestimmen können, ab welchem Suchstring ersetzt werden soll und wie viele Ersetzungen maximal vorgenommen werden sollen.

StringBuilder.Replace

Die `Replace`-Methode eines `StringBuilder`-Objekts erlaubt hingegen auch die Angabe einer Startposition und der Länge des Teilstrings, in dem gesucht werden soll. So können Sie z. B. einen sieben Zeichen langen Teilstring ab dem neunten Zeichen ersetzen:

```
string source = "AAA-aaa-AAA-aaa";
string find = "aaa";
string replacement = "999";
int start = 8;
int count = 7;
System.Text.StringBuilder sb = new System.Text.StringBuilder(source);
string result = sb.Replace(find, replacement, start, count).ToString();
```

Dieses Beispiel ergibt den String "AAA-aaa-AAA-999". Leider besteht hier ebenfalls keine Möglichkeit, anzugeben, dass der Vergleich case-insensitiv erfolgen soll. Außerdem ist diese Variante der Ersetzung wohl eher praxisfern, da in der Regel eher angegeben wird, ab welchem gefundenen Teilstring ersetzt und wie viele Teilstrings ersetzt werden.

Achtung

Entgegen der sonstigen Vorgehensweise im .NET Framework gibt die `Replace`-Methode eines `StringBuilder`-Objekts nicht nur den geänderten String zurück, sondern ändert diesen auch in seinem eigenen Speicher. Nach dem Ersetzen ergibt `ToString` also den geänderten String.

Regex.Replace

Die `Replace`-Methode eines regulären Ausdrucks in Form einer Instanz der `Regex`-Klasse erlaubt zudem das Ersetzen ohne Berücksichtigung der Groß-/Kleinschreibung. Am Argument `count` wird hier, anders als beim `StringBuilder`-Objekt, angegeben, wie viele Ersetzungen maximal erfolgen sollen.

```
string source = " AAA-aaa-AAA-aaa";
string find = "aaa";
string replacement = "999";
int start = 0;
int count = 2;
Regex re = new Regex(find, RegexOptions.IgnoreCase);
string result = re.Replace(source, replacement, count, start);
```

Das Ergebnis der `Replace`-Methode ist in diesem Fall "999-999-AAA-aaa", da die Groß-/Kleinschreibung nicht berücksichtigt wurde und zwei Ersetzungen erfolgt sind. Wenn Sie am dritten Argument -1 übergeben, werden alle Fundstellen ersetzt. Das letzte Argument der im Beispiel verwendeten Variante der `Replace`-Methode definiert übrigens die Zeichenposition, ab der gesucht werden soll. Leider können Sie hier nicht definieren, ab welcher Fundstelle ersetzt werden soll.

Abgesehen davon besitzt die `Replace`-Methode eines `Regex`-Objekts aber zwei gravierende Nachteile: Zum einen müssen Sie dafür sorgen, dass das Suchmuster korrekt ist. Reguläre Ausdrücke verwenden den Backslash als Escape-Zeichen und einige andere Sonderzeichen wie z. B. geschweifte und runde Klammern. Wenn diese Sonderzeichen in Ihrem Suchstring vorkommen, müssen Sie einen Backslash vor das jeweilige Zeichen setzen, um die Sonderbedeutung aufzu-

heben. Dass Sie reguläre Ausdrücke zur Suche einsetzen können, ist aber natürlich auch in einigen Fällen ein wichtiger Vorteil.

Zum anderen ist die Ausführung der `Regex.Replace`-Methode gegenüber den anderen Möglichkeiten recht langsam. In einem Test benötigte diese Methode für 10.000 Ersetzungen etwa 0,17 Sekunden, die `Replace`-Methode einer `string`-Instanz hingegen nur ca. 0,024 Sekunden.

Visual Basic Replace

Eine weitere Variante zum Ersetzen von Strings ist die Verwendung der Visual-Basic-`Replace`-Methode. Zur Verwendung dieser Methode müssen Sie die Assembly *Microsoft.VisualBasic.dll* referenzieren. In der Verweisliste von Visual Studio wird diese Assembly unter dem Namen *Microsoft Visual Basic* aufgeführt. Die `Replace`-Methode finden Sie dann im Modul `Strings` im Namensraum `Microsoft.VisualBasic`.

Am ersten Argument übergeben Sie den Quellstring, am zweiten den Suchstring und am dritten den Ersatzstring. Das vierte Argument gibt die Position an, ab der im Quellstring gesucht wird. Der Index beginnt hier anders als in C# mit 1. Dummerweise gibt die VB-`Replace`-Methode den String ab der Startposition zurück (und nicht komplett), wenn Sie eine Startposition größer 1 angeben (was in meinen Augen ein alter Visual-Basic-Bug ist).

Dafür können Sie am fünften Argument bestimmen, wie viele Ersetzungen maximal vorgenommen werden sollen. Wenn Sie hier -1 angeben, werden alle Vorkommen des Suchstrings ab der Startposition ersetzt.

Das letzte Argument steuert, ob die Groß-/Kleinschreibung berücksichtigt werden soll. Hier können Sie die Werte der `CompareMethod`-Aufzählung angeben. Der Wert `Binary` führt zu einem Vergleich unter Berücksichtigung der Groß-/Kleinschreibung, der Wert `Text` zu einem Textvergleich. Das folgende Beispiel ersetzt in einem String maximal zwei Vorkommen eines Teilstrings ohne Berücksichtigung der Groß-/Kleinschreibung:

```
string source = " AAA-aaa-AAA-aaa";
string find = "aaa";
string replacement = "999";
int start = 1;
int count = 2;
result = Strings.Replace(source, find, replacement, start, count,
   CompareMethod.Text);
```

Das Ergebnis des `Replace`-Aufrufs ist hier ” 999-999-AAA-aaa”.

Performance-Vergleich

Ich habe natürlich (mit Hilfe der Klasse `Stopwatch` aus dem Namensraum `System.Diagnostics`, siehe Rezept 74 Performance-Vergleiche durchgeführt um meine Version der `Replace`-Methode abzusichern. Tabelle 4 zeigt die Ergebnisse für 10.000 Aufrufe auf einem Rechner mit 2,4 GHz Prozessor, 512 MB Arbeitsspeicher und Windows XP. Zur Messung habe ich das Programm ohne Debuginformationen kompiliert und direkt (ohne Visual Studio) aufgerufen. Das Programm finden Sie in den Beispielen zu diesem Kapitel neben diesem Rezept.

Methode	Benötigte Zeit für 10.000 Aufrufe in Sekunden
`String.Replace`	0,0106468140503891
`StringBuilder.Replace`	0,0107597473980632
`Regex.Replace` mit binärem Vergleich	0,0707475391425447
`Regex.Replace` mit Textvergleich	0,0851926711355773
Visual-Basic-`Replace` mit binärem Vergleich	0,0342659472083742
Visual-Basic-`Replace` mit Textvergleich	0,0494198983390347
Eigene `Replace`-Methode mit binärem Vergleich	0,0563040833402011
Eigene `Replace`-Methode mit Textvergleich	0,0640651954368502

Tabelle 4: Performancevergleich der verschiedenen Replace-Methoden

Der absolute Gewinner ist die `Replace`-Methode der `String`-Klasse. Immer wenn es auf Performance ankommt und Sie die erweiterten Möglichkeiten der anderen Methoden nicht benötigen, sollten Sie diese Methode anwenden.

Die Verwendung eines `StringBuilder`-Objekts ist trotz der in der Schleife vorgenommenen Instanzierung (um einen korrekten Vergleich zu erhalten habe ich das `StringBuilder`-Objekt immer wieder neu instanziert) erstaunlich performant.

Die `Regex`-Variante ist deutlich langsamer als die anderen und sollte nur eingesetzt werden, wenn Sie zum Vergleich einen regulären Ausdruck einsetzen müssen.

Was mich erstaunte, war allerdings die Tatsache, dass meine `Replace`-Methode mit der von Visual Basic ganz gut mithalten kann. Es besteht also eigentlich kein Grund, die Visual-Basic-Variante einzusetzen ☺.

24 Strings an vorgegebenen Trennzeichen auftrennen

Strings können Sie recht einfach über die `Split`-Methode an einem oder mehreren vorgegebenen Trennzeichen auftrennen. `Split` gibt ein String-Array zurück, das die Teilstrings verwaltet:

```
string names = "Zaphod;Ford;Trillian";
string[] nameList = names.Split(';');
foreach (string name in nameList)
{
   Console.WriteLine(name);
}
```

Enthält der String mehrere unterschiedliche Trennzeichen, geben Sie diese in einzelnen Argumenten an:

```
string names = "Zaphod;Ford,Trillian";
string[] nameList = names.Split(';', ',');
foreach (string name in nameList)
{
   Console.WriteLine(name);
}
```

Leider ermöglicht `Split` nicht das Auftrennen von Strings, bei denen nicht einzelne Zeichen, sondern Strings die Teilstrings trennen. Das ist bereits dann der Fall, wenn ein String mehrere Zeilen enthält, die durch "\r\n" voneinander getrennt sind. Zum Auftrennen solcher Strings können Sie aber die `Split`-Methode der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) verwenden. Am ersten Argument übergeben Sie den aufzutrennenden String, am zweiten den Trennstring:

```
string rows = "Zeile 1\r\nZeile 2\r\nZeile 3";
string[] rowList = Regex.Split(rows, "\r\n");
foreach (string row in rowList)
{
   Console.WriteLine(row);
}
```

Dabei sollten Sie beachten, dass der Trennstring ein Muster der regulären Ausdrücke ist. In diesem besitzen einige Zeichen eine Sonderbedeutung. Ein Stern steht z. B. dafür, dass das links stehende Zeichen (oder die links stehende Gruppe) beliebig oft vorkommen kann. Die Bedeutung der Sonderzeichen können Sie aufheben, indem Sie diesen einen Backslash voranstellen. Informationen über reguläre Ausdrücke finden Sie an der Adresse *www.juergen-bayer.net/artikel/CSharp/Regulaere-Ausdruecke/Regulaere-Ausdruecke.aspx*.

25 Mehrere (Leer-)Zeichen in ein (Leer-)Zeichen konvertieren

Strings enthalten manchmal mehrere Leerzeichen, was besonders dann gilt, wenn die Eingaben von einem Anwender stammen. In vielen Fällen sind diese mehrfachen Leerzeichen störend und müssen durch ein Leerzeichen ersetzt werden. Diese Aufgabe ist einfach zu lösen. Ich habe einige Varianten der Programmierung verglichen (u. a. eine eigene Programmierung, die eine vereinfachte Variante der `Replace`-Methode aus dem Rezept 23 darstellte, und eine Variante, die mit einem regulären Ausdruck arbeitete). Die schnellste und zuverlässigste Variante war gleichzeitig auch die einfachste: die `ToSingleSpace`-Methode in Listing 29. Diese Methode ruft in einer Schleife lediglich die `Replace`-Methode des `string`-Objekts so lange auf, bis keine doppelten Leerzeichen mehr gefunden werden. Um noch ein wenig mehr Flexibilität zu besitzen, habe ich zudem die (nur unwesentlich langsamere) Methode `ToSingleChar` entwickelt, der Sie das Zeichen übergeben können, dessen mehrfaches Vorkommen in ein Vorkommen konvertiert werden soll. `ToSingleChar` erzeugt dazu einfach die zwei Stringvariablen `find` und `replace` mit zwei bzw. einem der übergebenen Zeichen und ruft damit die `Replace`-Methode des `string`-Objekts auf.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` importieren.

```
public static string ToSingleSpace(this string source)
{
   while (source.IndexOf("  ") > -1)
   {
      source = source.Replace("  ", " ");
```

Listing 29: Methoden zum Ersetzen mehrfacher Leerzeichen und anderer Zeichen

```
   }

   return source;
}

public static string ToSingleChar(this string source, char c)
{
   string find = new string(c, 2);
   string replace = new string(c, 1);
   while (source.IndexOf(find) > -1)
   {
      source = source.Replace(find, replace);
   }

   return source;
}
```

Listing 29: Methoden zum Ersetzen mehrfacher Leerzeichen und anderer Zeichen (Forts.)

26 Erstes Zeichen aller Wörter im String groß- und die restlichen Zeichen kleinschreiben

Um Strings in eine normalisierte Form zu bringen, ist es manchmal notwendig, die ersten Zeichen aller Wörter im String groß- und die folgenden kleinzuschreiben. Damit können Sie die oft unterschiedlichen Eingaben von Anwendern einigermaßen in eine gleichmäßige Form bringen. Wenn Sie dann noch mehrfache Leerzeichen entfernen, wie ich es im Rezept 25 beschreibe, erhalten Sie einen recht gut formatierten String.

Bei der vorliegenden Aufgabe ist das Problem eigentlich nur, die einzelnen Wörter zu extrahieren. Sie könnten dazu die `Split`-Methode eines Strings verwenden. Da Sie dieser Methode ein Zeichen-Array übergeben können, können Sie die unterschiedlichen Wort-Trennzeichen wie z. B. ein Leerzeichen oder einen Bindestrich berücksichtigen. Problematisch ist daran allerdings das korrekte Wieder-Zusammensetzen des Strings (mit den richtigen Wort-Trennzeichen).

Wenn Sie nur die ersten Zeichen jedes Worts im String großschreiben wollen, können Sie die Methode `TextInfo.ToTitleCase` eines `CultureInfo`-Objekts verwenden. Diese Methode konvertiert die einzelnen Wörter eines Strings in eine Kombination aus großgeschriebenem Anfangsbuchstaben und kleingeschriebenem Rest. Ausnahmen sind Wörter, die komplett in Großbuchstaben geschrieben sind. Diese bleiben in ihrer Schreibweise erhalten.

Die Konvertierung erfolgt hierbei kulturabhängig. Wenn Sie die aktuelle Kultur berücksichtigen wollen, können Sie die `CurrentCulture`-Eigenschaft der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) verwenden:

```
string source = "das ist ein TEST-sTRING";
string result = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(source);
```

Dieses Beispiel ergibt (auf einem Rechner, der die deutsche Kultur verwendet) den String "Das Ist Ein TEST-String".

Wenn Sie erreichen wollen, dass alle Wörter konvertiert werden, wandeln Sie den String einfach in die Kleinschreibung um:

```
string source = "das ist ein TEST-sTRING";
string result =
   CultureInfo.CurrentCulture.TextInfo.ToTitleCase(source.ToLower());
```

Dieses Beispiel ergibt (wieder auf einem Rechner, der die deutsche Kultur verwendet) den String "Das Ist Ein Test-String".

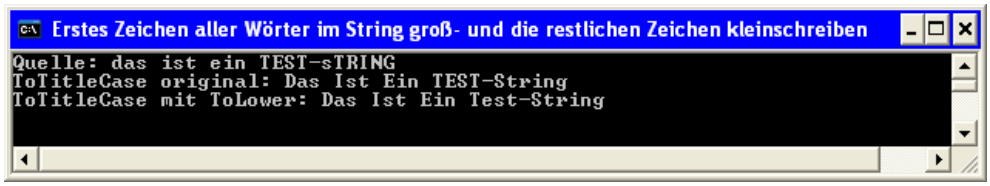


Abbildung 22: Ein Beispiel-Programmm, das ToTitleCase einsetzt

27 Anzahl der Wörter in einem String ermitteln

Die Anzahl der Wörter in einem String können Sie über den regulären Ausdruck \w{1,} ermitteln. \w steht hier für ein beliebiges Wort-Zeichen (Buchstaben, Zahlen und der Unterstrich). {1,} steht dafür, dass das zuvor definierte Zeichen mindestens einmal hintereinander vorkommen muss. Gegenüber einer eigenen Programmierung bietet die Verwendung eines regulären Ausdrucks den Vorteil, dass er sehr einfach und auch in Bezug auf die Unicode-Zeichen der Sprachen mit einer anderen als der lateinischen Schriftsprache (Griechisch, Kyrillisch etc.) sicher ist.

Die Methode `WordCount` in Listing 30 setzt dies um. Das Suchmuster wird neben dem Quellstring der statischen `Matches`-Methode der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) übergeben. Diese Methode gibt eine Referenz auf ein `MatchCollection`-Objekt zurück, das für alle Fundstellen im Quellstring `Match`-Objekte verwaltet. Diese Objekte interessieren uns für unsere Aufgabenstellung nicht. `WordCount` liest einfach die `Count`-Eigenschaft aus und gibt damit die Anzahl der Treffer zurück.

```
public static long WordCount(this string source)
{
   return Regex.Matches(source, @"\w{1,}").Count;
}
```

Listing 30: Sichere (und langsame) Methode zur Ermittlung der Anzahl der Wörter in einem String

Die Verwendung des regulären Ausdrucks und der `MatchCollection` führt dazu, dass die Methode relativ langsam ausgeführt wird (was besonders beim Überprüfen von Strings mit mehreren Kilobyte Zeichen negativ auffällt) und dass viel Speicher benötigt wird (die `Match`-Objekte werden ja schließlich angelegt).

Deshalb habe ich eine weitere Methode `WordCountLatin` geschrieben. Diese Methode geht den String Zeichen für Zeichen durch und überprüft, ob es sich um ein Nicht-Wort-Zeichen handelt.

Strings werden in einem .NET-Programm als 16-Bit-Unicode-Zeichenketten gespeichert[2]. Unicode ist aufgeteilt in einen allgemeinen, lateinischen Bereich und folgende sprachspezifische (z. B. Griechisch und Kyrillisch), Symbol-, mathematische und andere Teiltabellen. Diese Teiltabellen finden Sie auf der Unicode-Webseite an der Adresse *www.unicode.org/charts*.

`WordCountLatin` vergleicht den Unicode-Wert jedes Zeichens mit den Werten, die ich als Wort-Zeichen im lateinischen Bereich der Unicode-Tabelle erkannt habe. Unterstützt wird aufgrund der Performance und der Komplexität nur der Bereich bis zum Zeichen 0x2FF (lateinische Zeichen, IPA-Erweiterungen und Abstands-Modifizier-Zeichen).

Mit Hilfe der Tabelle *DerivedCoreProperties* der offiziellen Unicode-Datenbank (*www.unicode.org/Public/UNIDATA/DerivedCoreProperties.txt*) habe ich dazu zunächst die alphabetischen Zeichen ermittelt und den Zahlbereich und den Unterstrich hinzugefügt.

Dabei sind die folgenden Bereiche herausgekommen:

- Zeichen im Bereich von 0x0030 und 0x0039: Ziffern,
- Zeichen im Bereich von 0x0041 und 0x005A: Normale Großbuchstaben,
- das Zeichen 0x005F: Unterstrich,
- Zeichen im Bereich von 0x0061 bis 0x007A: Kleinbuchstaben,
- die Zeichen 0x00AA, 0x00B5 und 0x00BA: hochgestelltes, unterstrichenes a, Mü-Zeichen, hochgestelltes, unterstrichenes o,
- Zeichen im Bereich von 0x00C0 bis 0x00D6, 0x00D8 bis 0x00F6, 0x00F8 bis 0x0236: Länderspezifische Sonderzeichen,
- Zeichen im Bereich von 0x0250 bis 0x02C1, 0x02C6 bis 0x02D1, 0x02E0 bis 0x02E4 und das Zeichen 0x02EE: Wort-Zeichen der IPA-Erweiterung.

Wird das Zeichen als Wort-Zeichen erkannt und die Variable `newWord` ist `false`, beginnt mit diesem Zeichen ein neues Wort und `WordCountLatin` inkrementiert die Zählvariable `count` und setzt `newWord` auf `false`. Ist das Zeichen kein Wort-Zeichen, setzt `WordCountLatin` die Variable `newWord` auf `false`. Der Wert von `count` wird schließlich zurückgegeben. Zur Sicherheit wirft `WordCountLatin` eine Ausnahme, wenn der String ein Zeichen im nicht unterstützten Bereich enthält.

`WordCountLatin` erfordert den Import des Namensraums `System`.

2. Wenn Sie Unicode-Zeichen im Quellcode verwenden (z. B. über die Betätigung der Alt-Taste mit dem Unicode-Zeichencode im numerischen Tastaturblock), müssen Sie die Quellcodedatei explizit mit Codierung speichern, damit die Unicode-Zeichen in der originalen Codierung bestehen bleiben. Dazu wählen Sie den Befehl Speichern unter und klicken auf den kleinen Pfeil rechts am Speichern-Schalter. Wählen Sie dann den Befehl Mit Codierung speichern und im folgenden Dialog die Codierung Unicode (UTF-8 ohne Signatur) – Codepage 650001.

```
public static long WordCountLatin(this string source)
{
   bool newWord = false;
   long count = 0;

   // Alle Zeichen des Strings durchgehen
   for (int i = 0; i < source.Length; i++)
   {
      char c = source[i];

      // Überprüfen, ob das Zeichen größer ist als das letzte Zeichen im
      // unterstützten (lateinischen) Bereich der Unicode-Tabelle.
      if (c > '\u02FF')
         throw new Exception("WordCountLatin unterstützt keine Strings " +
            "mit Unicode-Zeichen größer 0x02FF");

      // Überprüfen, ob es ein Wort-Zeichen ist
      if ((c >= '\u0030' && c <= '\u0039') ||
         (c >= '\u0041' && c <= '\u005A') ||
         (c == '\u005F') ||
         (c >= '\u0061' && c <= '\u007A') ||
         (c == '\u00AA') ||
         (c == '\u00B5') ||
         (c == '\u00BA') ||
         (c >= '\u00C0' && c <= '\u00D6') ||
         (c >= '\u00D8' && c <= '\u00F6') ||
         (c >= '\u00F8' && c <= '\u0236') ||
         (c >= '\u0250' && c <= '\u02C1') ||
         (c >= '\u02C6' && c <= '\u02D1') ||
         (c >= '\u02E0' && c <= '\u02E4') ||
         (c == '\u02EE'))
      {
         // Wort-Zeichen
         if (newWord == false)
            count++;
         newWord = true;
      }
      else
      {
         // Kein Wort-Zeichen
         newWord = false;
      }
   }

   return count;
}
```

Listing 31: Schnelle und Speicher sparende Methode zum Zählen der Wörter in einem String

In einem Performancevergleich (der in das Beispielprogramm auf der Buch-CD integriert ist) habe ich für 10.000 Aufrufe mit dem String "ab cd ef" auf einem Pentium-4-Rechner mit 2,4 GHz für `WordCount` die (gerundete) Zeit 0,1375 Sekunden und für `WordCountLatin` die Zeit 0,0009208 Sekunden ermittelt. `WordCountLatin` ist also deutlich schneller.

28 Alle Wörter eines Strings extrahieren

Das Extrahieren aller Wörter aus einem String ist über den regulären Ausdruck \w{1,}, der alle Teile eines Strings ermittelt, die aus einem oder beliebig vielen Wort-Zeichen (Buchstaben, Zahlen und der Unterstrich) bestehen, recht einfach. Die Verwendung dieses Ausdrucks besitzt den Vorteil, dass die Wort-Zeichen aller in der Unicode-Zeichentabelle verwalteter Schriftsprachen korrekt erkannt werden. So können Sie z. B. auch griechische Texte korrekt auswerten.

Über die statische `Matches`-Methode der `Regex`-Klasse (aus dem Namensraum `System.Text.RegularExpressions`) erhalten Sie eine Referenz auf eine `MatchCollection`-Auflistung, die in Form von `Match`-Objekten alle Fundstellen im übergebenen String enthält, die dem ebenfalls übergebenen Muster entsprechen. Die Eigenschaft `Value` der `Match`-Objekte speichert den gefundenen Teilstring.

Die Methode `GetWords` in Listing 32 setzt diese Technik ein. `GetWords` kopiert die Werte der erhaltenen `Match`-Objekte in ein String-Array und gibt dieses zurück.

`GetWords` erfordert den Import der Namensräume `System` und `System.Text.RegularExpressions`.

```
public static string[] GetWords(this string source)
{
   // Alle Wörter abfragen
   MatchCollection matches = Regex.Matches(source, @"\w{1,}");

   // Die MatchCollection in ein String-Array kopieren und zurückgeben
   string[] words = new string[matches.Count];
   for (int i = 0; i < matches.Count; i++)
   {
      words[i] = matches[i].Value;
   }
   return words;
}
```

Listing 32: Methode zum Extrahieren aller Wörter aus einem String

Abbildung 23 zeigt das Ergebnis eines Test-Programms, bei dem der String »Das ist ein [Test-] String« zerlegt wurde.

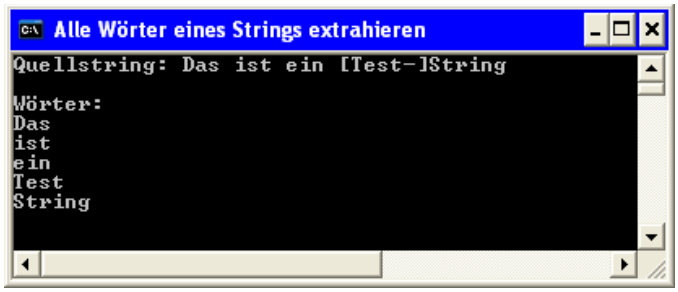


Abbildung 23: Ein Test-Programm hat den String korrekt zerlegt

29 Einen String an einem rechten Rand wortgerecht umbrechen

In Windows-Anwendungen sind die meisten Steuerelemente in der Lage, Text am rechten Rand automatisch wortgerecht umzubrechen. In manchen Fällen, in denen dieser Umbruch nicht automatisch geschieht, macht ein solcher Umbruch aber auch Sinn. Dies ist zum Beispiel der Fall, wenn Sie Text in einem `ToolTip` ausgeben und dafür sorgen wollen, dass diese Ausgabe nicht zu breit wird. Eine andere Anwendung wäre die auf eine stimmte Breite beschränkte Ausgabe von Text in einer Textdatei oder an der Konsole.

Hinweis

Die in diesem Rezept beschriebene Methode bricht den Text bezogen auf eine maximale Zeichenlänge um. In Rezept 348 finden Sie eine Methode, die Text bezogen auf eine maximale Pixel-Breite umbricht.

Die Grundidee beim Umbruch von Strings ist, dass diese zunächst in einzelne Zeilen umgewandelt und diese Zeilen dann in einzelne Wörter zerlegt werden. Beim Extrahieren der einzelnen Wörter müssen die möglichen Trennzeichen berücksichtigt und mit in das jeweilige Wort-Token aufgenommen werden. Die so erhaltene Auflistung von einzelnen Wort-Token können Sie dann durchgehen, die Länge der einzelnen Token addieren und mit einer maximalen Länge vergleichen. Ist die Maximallänge erreicht, generieren Sie einen Zeilenumbruch.

Die Methode `WordWrap` in Listing 33 arbeitet nach dieser Idee. Sie ersetzt zunächst Tabs im Text durch drei Leerzeichen um Probleme mit Tabulatoren zu vermeiden. Dann zerlegt `WordWrap` mithilfe der `Split`-Methode der `Regex`-Klasse den übergebenen Text in einzelne Zeilen und geht diese durch. Für die Verwaltung der resultierenden Zeilen setzt `WordWrap` einen `StringBuilder` (`wrappedText`) ein. Nach dem Anfügen eines Zeilenvorschubs an diesen `StringBuilder` wird der Text der aktuellen Zeile in einzelne Wort-Token gesplittet.

Dabei hatte ich das Problem, dass das Programm Wort-Trennzeichen erkennen musste. Relativ klar war, dass das Problem über einen regulären Ausdruck gelöst werden konnte. Leider bieten die regulären Ausdrücke in C# keine Zeichenklasse für Wort-Trennzeichen (nur die Zeichenklasse \b für Wort-Grenzen, aber diese stehen nicht für die Trennzeichen selbst, sondern für die Grenze zwischen dem Wort und dem Trennzeichen). Alle Wort-Trennzeichen zu ermitteln und in den regulären Ausdruck einzufügen wäre sehr aufwändig und fehleranfällig. Deswegen habe ich mich entschieden einfach alle Nicht-Wortzeichen und den Bindestrich (der eigentlich

ein Wortzeichen ist) als Trennzeichen zu verwenden. Daraus resultierte der reguläre Ausdruck *\w{1,}(\W|-){0,}*. \w{1,} steht für alle Wörter (siehe Rezept 28). (\W|-){0,} steht für Nicht-Wortzeichen oder Bindestriche in einer beliebigen Anzahl hintereinander.

Mit Hilfe dieses regulären Ausdrucks gelingt es `WordWrap` alle Wort-Token aus der aktuellen Zeile zu extrahieren. Der Rest ist einfach: In der Variablen `currentWidth` verwaltet `WordWrap` die aktuelle Breite. Beim Durchgehen der einzelnen Wort-Token wird zunächst überprüft, ob das erste Wort-Token bereits zu lang ist. Diesen Sonderfall musste ich behandeln, da ansonsten eine zusätzliche Leerzeile resultieren würde. Für alle anderen Token wird überprüft, ob die aktuelle Breite addiert mit der Länge des Token größer wäre als die maximale Breite. Ist dies nicht der Fall, wird das Token an den `StringBuilder wrappedText` angefügt. Im anderen Fall generiert `WordWrap` zusätzlich vorher über das Anfügen von `Environment.NewLine` an `wrappedText` einen Zeilenumbruch und setzt `currentWidth` zurück. `currentWidth` wird natürlich bei jedem Durchlauf um die Länge des aktuellen Wort-Token erhöht.

Schließlich gibt `WordWrap` das Ergebnis als String zurück.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Text.RegularExpressions` und `System.Text` importieren.

```
public static string WordWrap(this string text, int maxCharWidth)
{
   // StringBuilder für den umbrochenen Text erzeugen
   StringBuilder wrappedText = new StringBuilder();

   if (text != null && text.Length > 0)
   {
      // Tabs durch Leerzeichen ersetzen um
      // Darstellungs-Probleme zu vermeiden
      text = text.Replace("\t", "   ");

      // Aufteilen des Textes in einzelne Zeilen
      string[] rows = Regex.Split(text, Environment.NewLine);
      foreach (string row in rows)
      {
         // Zeilenumbruch an die vorherige Zeile anfügen
         if (wrappedText.Length > 0)
         {
            wrappedText.Append(Environment.NewLine);
         }

         // Text in einzelne Wort-Token aufsplitten
         // Als Trennzeichen werden alle Nicht-Wort-Zeichen und
         // der Bindestrich verwendet
         MatchCollection matches = Regex.Matches(row, @"\w{1,}(\W|-){0,}");
```

Listing 33: Methode zum wortgerechten Umbrechen von Strings bezogen auf eine maximale Zeichenlänge

```
            // Die einzelnen Wort-Token durchgehen und ausgeben
            int currentWidth = 0;
            for (int i = 0; i < matches.Count; i++)
            {
               Match match = matches[i];
               // Den Sonderfall behandeln, dass das erste Wort-Token
               // zu lang ist
               if (i == 0 && match.Value.Length > maxCharWidth)
               {
                  // Aktuelles Token ablegen
                  wrappedText.Append(match.Value);
               }
               else
               {
                  if (currentWidth + match.Value.Length <= maxCharWidth)
                  {
                     // Aktuelles Token ablegen
                     wrappedText.Append(match.Value);
                  }
                  else
                  {
                     // Das Wort passt nicht mehr in die Spalte
                     // Zeilenumbruch generieren
                     if (wrappedText.Length > 0)
                     {
                        wrappedText.Append(Environment.NewLine);
                     }

                     // Token ablegen
                     wrappedText.Append(match.Value);

                     // Die aktuelle Breite wieder auf den Anfang setzen
                     currentWidth = 0;
                  }

                  // Die aktuelle Breite um die Wortlänge hochzählen
                  currentWidth += match.Value.Length;
               }
            }
         }
      }

      // Ergebnis zurückgeben
      return wrappedText.ToString();
   }
```

Listing 33: Methode zum wortgerechten Umbrechen von Strings bezogen auf eine maximale Zeichenlänge (Forts.)

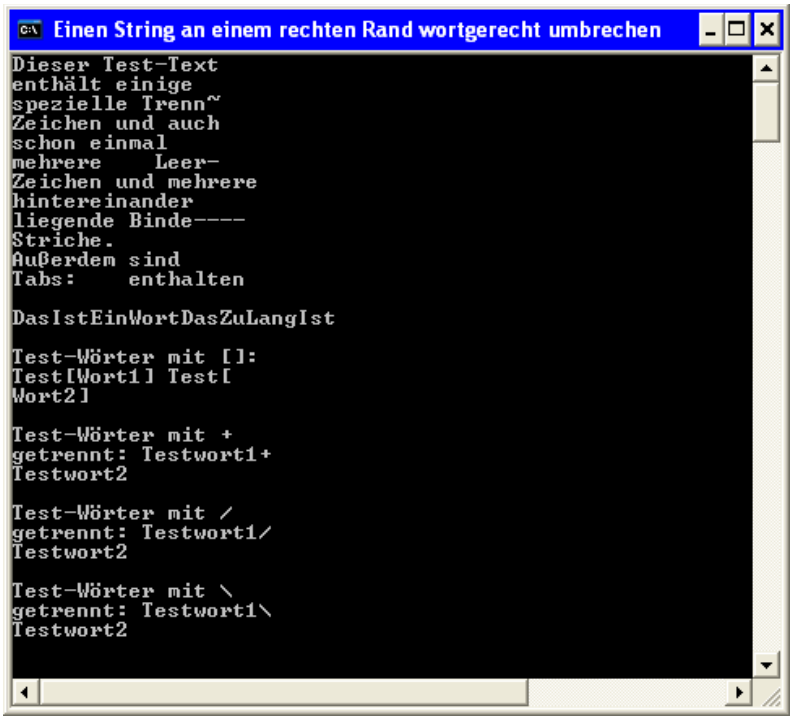


Abbildung 24: Ein wortgerecht umbrochener Text in einer Beispiel-Anwendung

Hinweis

`WordWrap` hat Probleme mit Wörtern, die länger sind als die maximale Zeichenlänge, wie Sie in Abbildung 24 sehen können. Solche Wörter werden natürlich über den gesetzten Rand hin ausgegeben. Dieses Problem ist nur sehr schwer lösbar, da Sie in diesem Fall eine automatische und grammatikalisch korrekte Wort-Trennung implementieren müssten.

30 String auf die ersten Wörter bis zu einer Maximallänge kürzen

Oftmals ist es sinnvoll, einen langen String zur Darstellung auf dem Bildschirm so einzukürzen, dass dieser in einem gegebenen Raum dargestellt werden kann. Ein Beispiel dafür sind Internet-Suchmaschinen wie Google, die eine verkürzte Form der Beschreibung der gefundenen Webseiten anzeigen. Dabei ist es aber nicht ideal, den String einfach zu kürzen, da dieser dann mit großer Wahrscheinlichkeit unschön mitten in einem Wort abgetrennt wird.

Die Methode `Abbreviate` in Listing 34 kürzt einen String so, dass dieser der gegebenen Maximallänge an Zeichen entspricht, aber nicht mitten in einem Wort abgeschnitten wird. Als Wort-Trennzeichen wird nur das Leerzeichen verwendet, um zu erreichen, dass nicht mitten in mit Bindestrichen zusammengesetzten Wörtern oder in Wörtern mit angehängten Klammern oder Ähnlichem gekürzt wird.

`Abbreviate` splittet den String einfach an den Leerzeichen auf und setzt die einzelnen Teilstrings in einer Schleife so lange zusammen, bis die maximal mögliche Länge erreicht ist. Bei der maximalen Länge werden die drei Punkte mit eingerechnet, die `Abbreviate` an einen gekürzten String anhängt. `Abbreviate` berücksichtigt aber auch die Sonderfälle, dass schon das erste Wort zu lang ist oder dass der String insgesamt kürzer ist als die Maximallänge.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System` einbinden.

```
public static string Abbreviate(this string source, int maxCharCount)
{
   string result = String.Empty;

   // String an Leerzeichen splitten
   string[] words = source.Split(' ');

   // Die Sonderfälle abhandeln, dass der gesamte String
   // kürzer oder das erste Wort schon zu lang ist
   if (source.Length <= maxCharCount)
   {
      return source;
   }
   if (words.Length > 0 && words[0].Length > maxCharCount)
   {
      return words[0].Substring(0, maxCharCount - 3) + "...";
   }

   // Die Wörter durchgehen und in das Ergebnis schreiben,
   // bis die Maximallänge erreicht ist
   for (int i = 0; i < words.Length; i++)
   {
      if (result.Length + words[i].Length + 4 > maxCharCount)
      {
         return result + "...";
      }
      else
      {
         result += ' ' + words[i];
      }
   }

   return source;
}
```

Listing 34: Methode zum Kürzen eines Strings unter Berücksichtigung der Wörter

Listing 35 zeigt einen beispielhaften Aufruf der `Abbreviate`-Methode.

```
string source = "Die Antwort auf die Frage aller Fragen ist 42";
string result = source.Abbreviate(20);
Console.WriteLine("Quellstring: {0}", source);
Console.WriteLine("Gekürzter String: {0}", result);

source = "Formular-Authentifizierung";
result = source.Abbreviate(20);
Console.WriteLine("Quellstring: {0}", source);
Console.WriteLine("Gekürzter String: {0}", result);
```

Listing 35: Beispiel-Aufruf der Abbreviate-Methode

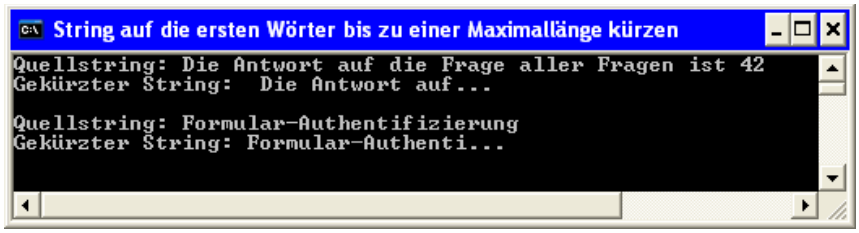


Abbildung 25: Wortgerechtes Kürzen eines Strings auf maximal 20 Zeichen

31 Teilstring nur dann ersetzen, wenn dieser am Anfang eines Strings gespeichert ist

Bei der Entwicklung einiger Methoden für dieses Buch musste ich einen Teilstring in einem String ersetzen, wenn dieser am Anfang des Strings gespeichert war. Ein Beispiel dafür ist das Rezept 140, das zwei Ordner auf einen identischen Inhalt vergleicht.

Dieses Problem habe ich über einen regulären Ausdruck gelöst, der mit einem ^-Zeichen begann und mit der gesuchten Zeichenkette endete. Das ^-Zeichen besagt, dass das folgende Muster am Anfang der Zeichenkette stehen muss. Um den am Anfang stehenden Teilstring zu ersetzen, rufen Sie die `Replace`-Methode der `Regex`-Klasse aus dem Namensraum `System.Text.RegularExpressions` auf. Dabei müssen Sie allerdings beachten, dass reguläre Ausdrücke einer Vielzahl an Zeichen eine besondere Bedeutung geben. Der Backslash wird z. B. als Einleitung einer Escape-Sequenz erkannt, runde Klammern werden für Gruppierungen verwendet.

Die Methode `ReplaceLeadingString` in Listing 36 stellt deswegen über die `Escape`-Methode der `Regex`-Klasse allen Sonderzeichen der regulären Ausdrücke im Suchstring einen Backslash voran, der die Bedeutung des Zeichens aufhebt.

`ReplaceLeadingString` ruft dann `Regex.Replace` auf, wobei der Aufruf entsprechend dem Argument `ignoreCase` entweder mit dem Wert `RegexOptions.IgnoreCase` am vierten Argument erfolgt oder ohne diesen Wert. Damit ist (als kleiner Zusatz) auch das Ersetzen ohne Berücksichtigung der Groß-/Kleinschreibung möglich.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Text.RegularExpressions` importieren.

```
public static string ReplaceLeadingString(this string source, string find,
   string replacement, bool ignoreCase)
{
   // Muster für die Suche zusammenstellen, dabei den Sonderzeichen im
   // Suchstring einen Backslash voranstellen
   string pattern = "^" + Regex.Escape(find);

   // Muster ersetzen
   if (ignoreCase)
   {
      return Regex.Replace(source, pattern, replacement,
         RegexOptions.IgnoreCase);
   }
   else
   {
      return Regex.Replace(source, pattern, replacement);
   }
}
```

Listing 36: Ersetzen eines Teilstrings am Anfang eines Strings

Dass `ReplaceLeadingString` wunderbar funktioniert, beweist das folgende Testprogramm, bei dem der Basis- und der Suchstring bewusst mit einigen Sonderzeichen der regulären Ausdrücke beginnen:

```
string source = @"^$ {}[] \1 \2 # \w. C:\Codebook\Basics\Strings (Test)";
string find = @"^$ {}[] \1 \2 # \w. C:\Codebook";
string replacement = @"C:\Books\C#-Codebook";
string result = source.ReplaceLeadingString(find, replacement, true);
```

Listing 37: Testprogramm für die ReplaceLeadingString-Methode

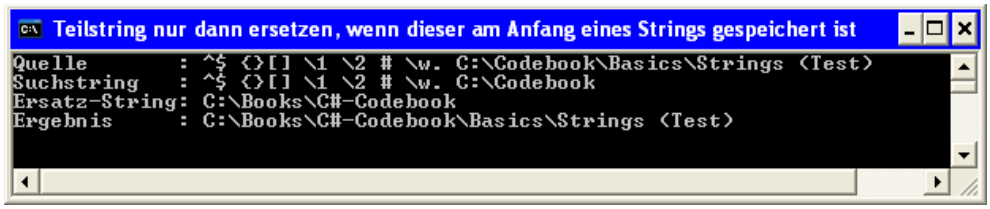


Abbildung 26: Die Ersetzung des Anfangs-Teilstrings war erfolgreich

32 Linken und rechten Teilstring extrahieren

Bei der String-Verarbeitung kommt es immer wieder vor, dass ein linker oder rechter Teilstring extrahiert werden muss. `string`-Objekte bieten zwar eine Methode `Substring`, die das Extrahieren beliebiger Teilstrings erlaubt. Das Auslesen des linken und mehr noch eines rechten Teilstrings ist damit aber für die Praxis ein wenig zu aufwändig. Hinzu kommt, dass `Substring` wenig tolerant ist und die Anfangs- und Endposition sich immer auf eine Position im String beziehen muss.

Wenn Sie nur einzelne String-Extraktionen programmieren und die Performance der Extraktion keine Rolle spielt, können Sie dazu auch die Methoden `Left` und `Right` in Listing 38 verwenden. Diese Methoden erlauben einen einfacheren Aufruf und arbeiten toleranter als `Substring`. Wenn Sie an `count` eine Anzahl Zeichen übergeben, die größer ist als der String selbst, geben beide Methoden einfach den übergebenen String unverändert zurück.

Zum Kompilieren dieser Methoden müssen Sie den Namensraum `System` importieren.

```
/* Methode zum Extrahieren eines linken Teilstrings */
public static string Left(this string source, int count)
{
   if (source.Length >= count)
   {
      return source.Substring(0, count);
   }
   else
   {
      return source;
   }
}

/* Methode zum Extrahieren eines rechten Teilstrings */
public static string Right(this string source, int count)
{
   int length = source.Length;
   if (length >= count)
   {
      return source.Substring(length - count, count);
   }
   else
   {
      return source;
   }
}
```

Listing 38: Methoden zum Extrahieren des linken und rechten Teilstrings

Die Performance beider Methoden ist dennoch nicht schlecht. In Tabelle 5 finden Sie einen Vergleich mit `Substring`.

Methode	Zeit für 10.000 Aufrufe in Sekunden (Pentium 4, 2,4 GHz)
Left	0,00112025411050846
`Substring`(0, *count*)	0,000840050900323924
Right	0,00112137157096782
`Substring`(`Length` – *count*, *count*)	0,000844520742161364

Tabelle 5: Performance der Left- und Right-Methoden im Vergleich mit einem direkten Substring-Aufruf

Den Performancetest finden Sie im Beispielprogramm zu diesem Rezept auf der CD.

33 Zahlen aus einem String extrahieren

Bei der Verarbeitung von Strings kommt es vor, dass Zahlen, die im String neben anderen Zeichenfolgen gespeichert sind, extrahiert werden müssen. Das ist z. B. dann der Fall, wenn Sie eine eingegebene Telefonnummer normalisieren müssen. Anwender geben Telefonnummern in verschiedenen Formaten ein, wobei die einzelnen Nummernteile meist durch Leerzeichen, Bindestriche oder Klammern getrennt werden. Eine solche Eingabe können Sie so zerlegen, dass nur die eingegebenen Zahlen herauskommen, und diese mit vordefinierten Trennzeichen dann wieder zusammensetzen.

Wenn ein String eine oder mehrere Zahlen enthält und zwischen diesen beliebige Zeichen gespeichert sind, können Sie diese Zahlen recht einfach über einen regulären Ausdruck extrahieren. Dazu können Sie die `Matches`-Methode der `Regex`-Klasse aus dem Namensraum `System.Text.RegularExpressions` verwenden. Dieser Methode übergeben Sie am ersten Argument den Quellstring und am zweiten das Suchmuster. Zum Suchen aller Ganzzahlen können Sie das Muster \d{1,} verwenden. Dieses Muster bedeutet, dass eine Folge von Ziffern mit beliebig vielen Ziffern gesucht wird. Wenn Sie auch Zahlen suchen, die einen Nachkommaanteil besitzen, können Sie das folgende Muster verwenden: \d{1,},{0,1}\d{0,}. Das Komma hinter \d{1,} muss natürlich durch das oder die Dezimaltrennzeichen ersetzt werden, die in der aktuellen Kultur verwendet werden.

`Matches` gibt eine Referenz auf eine Instanz der Klasse `MatchCollection` zurück, die eine Auflistung von `Match`-Objekten ist. Aus der Eigenschaft `Value` dieser Objekte können Sie den Teilstring auslesen, der dem Muster entsprach.

Die Methode `ExtractNumbers` im folgenden Listing geht auf diese Weise vor um alle Zahlen aus einem String in ein `double`-Array zu kopieren. Wenn Zahlen mit Nachkommaanteil extrahiert werden sollen (was am zweiten Argument erkannt wird), werden die aktuellen Dezimaltrennzeichen ausgelesen und über `Regex.Escape` so umgewandelt, dass alle eventuell vorhandenen Metazeichen für reguläre Ausdrücke über einen Backslash maskiert werden.

```
public static double[] ExtractNumbers(this string source,
   bool extractOnlyIntegers)
{
```

Listing 39: Methode zum Extrahieren aller Ganzzahlen aus einem String

```
   // Muster für den regulären Ausdruck definieren
   string pattern;
   if (extractOnlyIntegers)
   {
      pattern = @"\d{1,}";
   }
   else
   {
      string decimalSeparator = Regex.Escape(
         Thread.CurrentThread.CurrentCulture.NumberFormat.
         NumberDecimalSeparator);
      pattern = @"\d{1,}" + decimalSeparator + @"{0,1}\d{0,}";
   }

   // Die Treffer ermitteln
   MatchCollection matches = Regex.Matches(source, pattern);

   // Das Ergebnis in ein double-Array kopieren
   double[] result = new double[matches.Count];
   for (int i = 0; i < matches.Count; i++)
   {
      result[i] = Convert.ToDouble(matches[i].Value);
   }

   return result;
}
```

Listing 39: Methode zum Extrahieren aller Ganzzahlen aus einem String (Forts.)

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Threading` und `System.Text.RegularExpressions` importieren.

34 Überprüfen, ob ein String mit einer Zahl beginnt

Wenn Sie überprüfen wollen, ob ein String mit einer Ganzzahl beginnt, können Sie ähnlich wie im vorhergehenden Rezept einen regulären Ausdruck einsetzen. Das Muster ^\d{1,} trifft nur für beliebige Ziffernfolgen zu, die am Anfang der Zeichenkette stehen, und findet damit Ganzzahlen. Über ^\d{1,},{0,}\d{0,} können Sie auch Zahlen mit Nachkommaanteil am Anfang eines Strings abfragen (wobei Sie wieder das Komma durch die aktuell verwendeten Dezimaltrennzeichen ersetzen müssen).

Über die `Match`-Methode der `Regex`-Klasse können Sie überprüfen, ob ein String dem Muster entspricht. Dieser Methode übergeben Sie am ersten Argument den Quellstring und am zweiten das Muster. `Match` gibt eine Referenz auf ein `Match`-Objekt zurück. Aus dessen Eigenschaft `Success` können Sie auslesen, ob ein Teilstring gefunden wurde, der dem Muster entspricht.

Die Methode `StartsWithNumber` in Listing 40 setzt diese Technik ein. `StartsWithNumber` gibt im Erfolgsfall die Länge des gefundenen Teilstrings zurück, die aus der Eigenschaft `Length` des

Match-Objekts ausgelesen wird. Beginnt der übergebene String nicht mit einer Zahl, gibt StartsWithNumber 0 zurück.

```
public static int StartsWithNumber(this string source,
   bool considerOnlyIntegers)
{
   // Muster für den regulären Ausdruck definieren
   string pattern;
   if (considerOnlyIntegers)
   {
      pattern = @"^\d{1,}";
   }
   else
   {
      string decimalSeparator = Regex.Escape(
         Thread.CurrentThread.CurrentCulture.NumberFormat.
         NumberDecimalSeparator);
      pattern = @"^\d{1,}" + decimalSeparator + @"{0,1}\d{0,}";
   }

   // Match-Instanz erzeugen und überprüfen, ob ein Treffer erzielt wurde
   Match match = Regex.Match(source, pattern);
   if (match.Success)
   {
      // Die Länge des gefundenen Teilstrings zurückgeben
      return match.Length;
    }
   else
   {
      return 0;
   }
}
```

Listing 40: Methode, die überprüft, ob ein String mit einer Zahl beginnt, und die die Zeichenlänge der Zahl zurückgibt

StartsWithNumber erfordert den Import der Namensräume System, System.Threading und System.Text.RegularExpressions.

Über die Rückgabe der Länge des Strings können Sie die Zahl später sehr einfach aus dem String extrahieren:

```
string source = "1234 abc";
int result = source.StartsWithNumber(true);
if (result > 0)
{
```

Listing 41: Ermitteln, ob ein String mit einer Zahl beginnt, und extrahieren dieser Zahl

```
   long number = Convert.ToInt64(source.Substring(0, result));
   Console.WriteLine("'{0}' beginnt mit der Ganzzahl {1}",
      source, number);
}
else
{
   Console.WriteLine("'{0}' beginnt nicht mit einer Ganzzahl", source);
}

source = "A1234 abc";
result = source.StartsWithNumber(true);
if (result > 0)
{
   long number = Convert.ToInt64(source.Substring(0, result));
   Console.WriteLine("'{0}' beginnt mit der Ganzzahl {1}", source, number);
}
else
{
   Console.WriteLine("'{0}' beginnt nicht mit einer Ganzzahl", source);
}

source = "1234,567 abc";
result = source.StartsWithNumber(false);
if (result > 0)
{
   double number = Convert.ToDouble(source.Substring(0, result));
   Console.WriteLine("'{0}' beginnt mit der Dezimalzahl {1}", source, number);
}
else
{
   Console.WriteLine("'{0}' beginnt nicht mit einer Dezimalzahl", source);
}
```

Listing 41: Ermitteln, ob ein String mit einer Zahl beginnt, und extrahieren dieser Zahl (Forts.)

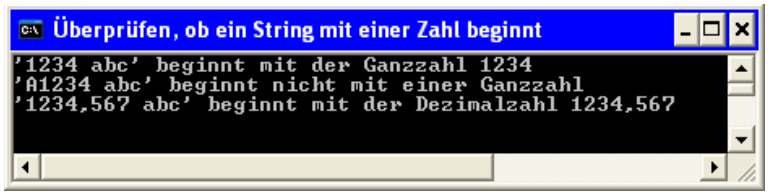


Abbildung 27: Das Beispielprogramm in Aktion

35 Zufalls-String berechnen

Die Berechnung einer zufälligen Zeichenkette ist besonders wichtig für automatisch generierte Passwörter, so wie diese von vielen Webseiten erzeugt werden, die eine Registration erfordern. Das Berechnen eines zufälligen Strings ist recht einfach.

Die Methode `GetRandomString` erzeugt dazu für jedes Zeichen des Strings eine Zufallszahl im Bereich der Unicode-Codes der normalen Kleinbuchstaben a bis z (97 bis 122) und hängt das jeweilige Zeichen an einen Ergebnisstring an.

Da die `Random`-Klasse keine echten Zufallszahlen erzeugt (siehe Rezept 18), verwendet `GetRandomString` eine Instanz der Klasse `RNGCryptoServiceProvider`, deren Methode `GetNonZeroBytes` »kryptografisch starke« Zufallszahlen im Bereich von 1 bis 255 erzeugt. Abhängig vom Argument `type` werden Zufallsstrings erzeugt, die aus allen »normalen« Zeichen zwischen dem ASCII-Wert 33 und 126, ausschließlich aus Buchstaben oder ausschließlich aus kleingeschriebenen Buchstaben bestehen.

Um für die Ermittlung eines Zufallsstrings mit allen normalen Zeichen Zahlen zwischen 33 und 126 zu erhalten, teilt `RandomString` jede ermittelte Zahl durch 2,713 (das entspricht etwa 255 / (126 – 33 + 1)), addiert 33 hinzu und wandelt das `double`-Ergebnis in einen `byte`-Wert um. Der berechnete Wert wird schließlich in ein Zeichen umgewandelt und an den Ergebnisstring angehängt. Für Zufallsstrings mit Buchstaben wird ähnlich vorgegangen, nur dass hier Zeichen zwischen den ASCII-Werten 97 und 122 ermittelt werden. Für den Fall, dass Großbuchstaben enthalten sein sollen, wird das ermittelte Zeichen (dieses Mal mit Hilfe der `Random`-Klasse) mehr oder weniger zufällig in einen Großbuchstaben umgewandelt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Text` und `System.Security.Cryptography` importieren.

```
/* Enum für die Art der Erzeugung von Zufallsstrings */
public enum RandomStringType
{
   /* Alle normalen Zeichen zwischen 33 und 122 */
   AllRegularChars,

   /* Nur Buchstaben */
   OnlyLetters,

   /* Nur kleingeschriebene Buchstaben */
   OnlyLowercaseLetters
}

/* Erzeugt einen Zufallsstring */
public static string GetRandomString(int count, RandomStringType type)
{
   StringBuilder randomString = new StringBuilder(count);

   // Echte Zufallszahlen im Bereich von 1 bis 255 erzeugen
   RNGCryptoServiceProvider rngCSP = new RNGCryptoServiceProvider();
   byte[] numbers = new byte[count];
   rngCSP.GetNonZeroBytes(numbers);
```

Listing 42: Methode zur Erzeugung eines Zufalls-Strings

```
   // Random-Instanz für einen String mit Großbuchstaben erzeugen
   Random ucaseRandom = new Random();

   if (type == RandomStringType.AllRegularChars)
   {
      // Die Zahlen so umrechnen, dass Werte zwischen 33 und 126
      // herauskommen, und die daraus resultierenden Zeichen an den
      // Ergebnisstring anhängen
      for (int i = 0; i < count; i++)
      {
         numbers[i] = (byte)(numbers[i] / 2.713 + 33);
         randomString.Append((char)numbers[i]);
      }
   }
   else
   {
      // Die Zahlen so umrechnen, dass Werte zwischen 97 und 122
      // herauskommen, und die daraus resultierenden Zeichen an den
      // Ergebnisstring anhängen
      for (int i = 0; i < count; i++)
      {
         numbers[i] = (byte)(numbers[i] / 9.81 + 97);
         if (type == RandomStringType.OnlyLetters)
         {
            // Den Buchstaben zufällig in einen Großbuchstaben umwandeln
            if (ucaseRandom.Next(3) == 2)
            {
               numbers[i] = (byte)(numbers[i] - 32);
            }
         }
         randomString.Append((char)numbers[i]);
      }
   }

   return randomString.ToString();
}
```

Listing 42: Methode zur Erzeugung eines Zufalls-Strings (Forts.)

36 Konstantennamen einer Aufzählung als String auslesen

In Anwendungen ist es häufig sinnvoll, die symbolischen Konstanten, die eine Aufzählung (Enumeration) zur Verfügung stellt, als String darzustellen. So können Sie dem Anwender z. B. in einer `ListBox` alle Werte einer Aufzählung zur Auswahl anbieten.

Die Klasse `Enum` bietet mit der statischen Methode `GetNames` eine einfache Möglichkeit, die Konstanten einer Aufzählung als String darzustellen. Dieser Methode übergeben Sie den Typ der Aufzählung und erhalten ein `string`-Array zurück.

Wenn Sie nur einzelne Konstantennamen auslesen wollen, können Sie die `GetName`-Methode verwenden, der Sie neben dem Typ der Aufzählung am zweiten Argument den Wert übergeben. Das macht allerdings nur dann Sinn, wenn Sie den Wert nicht über die symbolische Konstante, sondern direkt angeben.

Listing 43 zeigt die Konstantennamen für die Aufzählung `System.DayOfWeek`. Die Namen dieser Aufzählung werden in diesem Beispiel im `Load`-Ereignis eines Formulars in eine `ListBox` gelesen. Der Name der Konstante mit dem Wert 0 wird in ein `Label` ausgegeben.

```
private void StartForm_Load(object sender, System.EventArgs e)
{
   // Alle Werte einer Aufzählung als String auslesen
   string[] enumNames = Enum.GetNames(typeof(DayOfWeek));
   for (int i = 0; i < enumNames.Length; i++)
   {
      this.daysListBox.Items.Add(enumNames[i]);
   }

   // Einen bestimmten Wert auslesen
   string name = Enum.GetName(typeof(DayOfWeek), 0);
   this.infoLabel.Text = "Der Name der Konstante mit dem Wert 0 ist " + name;
}
```

Listing 43: Auslesen der Konstantennamen einer Aufzählung

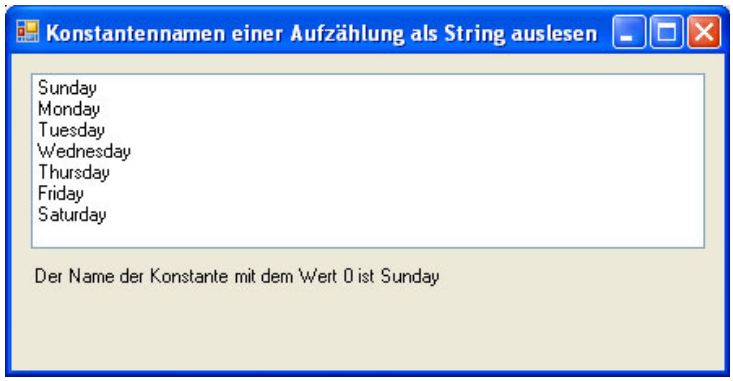


Abbildung 28: Das Programm hat die Konstantennamen der DayOfWeek-Aufzählung ausgelesen

37 Strings in Aufzählungswerte konvertieren

Im Rezept 36 habe ich gezeigt, wie Sie die Konstantennamen einer Aufzählung als String auslesen können um diese dem Anwender zur Auswahl anbieten zu können. Etwas problematisch ist allerdings die Auswahl der Anwender-Selektion. Als Problemlösung können Sie in der `ListBox` statt einfachen Strings eigene Objekte für die einzelnen Konstantenwerte verwalten, deren `ToString`-Methode den Namen der Konstante zurückgibt (sodass dieser angezeigt wird)

und die in einer separaten Eigenschaft den Wert der Konstante verwalten. Wenn Ihnen das zu aufwändig ist, können Sie aber auch einfach die `Parse`-Methode der `Enum`-Klasse verwenden. Dieser Methode übergeben Sie den Typ der Aufzählung und den String, den Sie umwandeln wollen. Am dritten Argument können Sie mit `true` bestimmen, dass die Groß-/Kleinschreibung nicht berücksichtigt werden soll.

`Parse` gibt den Aufzählungswert als `object` zurück, Sie müssen die Rückgabe also entsprechend umwandeln. Natürlich erzeugt `Parse` eine Ausnahme, wenn der übergebene String keinem Konstantennamen entspricht. Diese Ausnahme sollten Sie abfangen, wenn der String vom Anwender eingegeben wurde.

Zum Kompilieren des Beispiels müssen Sie den Namensraum `System` importieren.

```
string value = "friday";
try
{
   // Konstantenwert auslesen und in den Typ DayOfWeek casten
   DayOfWeek day = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), value, true);

   Console.WriteLine("Wert von '{0}': {1}", value, (int)day);
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 44: Umwandeln eines Strings in einen Aufzählungswert

38 String in Byte-Array umwandeln

Einige spezielle Techniken des .NET Framework arbeiten mit Byte-Arrays, die die einzelnen Zeichen eines Strings verwalten. Dazu gehören z. B. einige Klassen des Namensraums `System.Security.Cryptography`, über deren Methoden Sie Daten ver- und entschlüsseln können, und die Klasse `System.IO.MemoryStream`, die einen Stream im Arbeitsspeicher erzeugt (und die für einige Tricks in diesem Buch eingesetzt wird).

Einen String können Sie nun sehr einfach in ein Byte-Array konvertieren. Dazu verwenden Sie die `GetBytes`-Methode eines `Encoding`-Objekts, der Sie in einer Variante einen String übergeben können. Diese Methode setzt die einzelnen Zeichen des übergebenen Strings in deren zur Codierung passenden Byte-Werte um. Sie müssen nun nur noch die richtige Codierung herausfinden und ein entsprechendes `Encoding`-Objekt erzeugen. Wenn Sie, was wohl für die meisten Anwendungsfälle ausreicht, die Standard-8-Bit-Codierung ISO-8859-1 (ISO Latin 1) verwenden wollen, erzeugen Sie dieses Objekt über die statische Methode `GetEncoding` der `Encoding`-Klasse, der Sie den Namen der Codierung übergeben. Listing 45 zeigt, wie Sie aus einem String auf diese Weise ein Byte-Array erzeugen. Das Programm benötigt den Import der Namensräume `System` und `System.Text`.

```
// String in Byte-Array in der ISO-8859-1-Codierung umwandeln
string source = "abc-äöü";
Console.WriteLine("String in Byte-Array in der " +
   "ISO-8859-1-Codierung umwandeln");
Console.WriteLine();
Console.WriteLine("Originaler String: {0}", source);
byte[] result = System.Text.Encoding.GetEncoding(
   "ISO-8859-1").GetBytes(source);

// Byte-Array testweise ausgeben
Console.WriteLine("Das Byte-Array:");
for (int i = 0; i < result.Length; i++)
{
   Console.WriteLine("{0}: {1} ", result[i], (char)result[i]);
}

// String zurückkonvertieren
source = System.Text.Encoding.GetEncoding(
   "ISO-8859-1").GetString(result);
Console.WriteLine("Zurückkonvertierter String: {0}", source);
```

Listing 45: Umwandeln eines Strings in ein Byte-Array in der Codierung ISO-8859-1

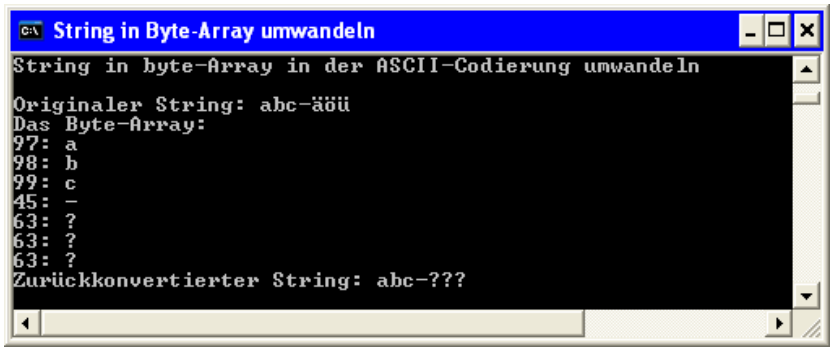


Abbildung 29: Das Programm hat die Zeichen des Teststrings korrekt in Bytes umgewandelt

Sie müssen bei der Umwandlung allerdings eines beachten: Strings werden in C# intern als 16-Bit-Unicode-Zeichen gespeichert. Wurde der String z. B. aus einer Unicode-Textdatei eingelesen, kann es sein, dass Zeichen gespeichert sind, deren Wert größer ist als 0x00FF (der größte 8-Bit-Wert). Eine Konvertierung mit einer 8-Bit-Codierung führt dann dazu, dass alle für diese Codierung unbekannten Zeichen in ein Fragezeichen (mit dem Code 0x003F) konvertiert werden. Dasselbe gilt natürlich auch, wenn Sie einen String, der lediglich 8-Bit-Sonderzeichen wie unsere Umlaute enthält, in die 7-Bit-ASCII-Codierung umwandeln (z. B. über das `Encoding`-Objekt, das Sie über die `ASCII`-Eigenschaft der `Encoding`-Klasse erreichen). Eine Rück-Umwandlung des Byte-Arrays führt dann natürlich zu einem anderen String als dem ursprünglichen. Das Beispiel in Listing 46 zeigt, was ich meine.

```
// String in Byte-Array in der ASCII-Codierung umwandeln
string source = "abc-äöü";
Console.WriteLine("String in Byte-Array in der ASCII-Codierung umwandeln");
Console.WriteLine();
Console.WriteLine("Originaler String: {0}", source);
byte[] result = System.Text.Encoding.ASCII.GetBytes(source);

// Byte-Array testweise ausgeben
Console.WriteLine("Das Byte-Array:");
for (int i = 0; i < result.Length; i++)
{
   Console.WriteLine("{0}: {1} ", result[i], (char)result[i]);
}

// String zurückkonvertieren
source = System.Text.Encoding.ASCII.GetString(result);
Console.WriteLine("Zurückkonvertierter String: {0}", source);
```

Listing 46: Umwandeln eines Unicode-Strings mit Zeichen größer 0x00fF (dem größten 7-Bit-ASCII-Wert) in ein Byte-Array und wieder zurück

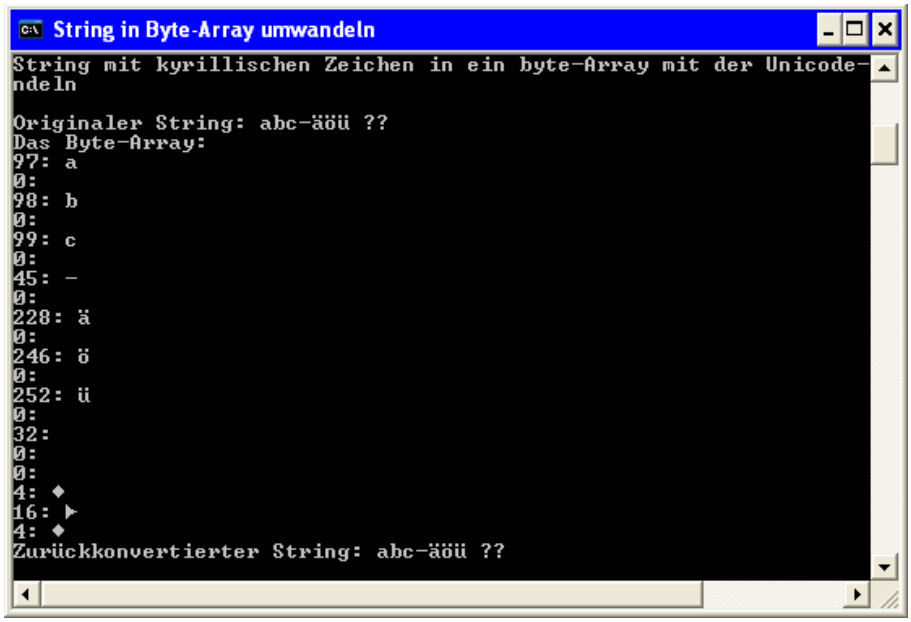


Abbildung 30: Die Umlaute wurden in ein Fragezeichen umgewandelt

Wenn Sie Strings mit Zeichen größer als 0x00FF umwandeln wollen, müssen Sie die Unicode-Codierung verwenden:

```
// String in Byte-Array in der Unicode-Codierung umwandeln
string source = "abc-äöü \u0400\u0410";
Console.WriteLine("String mit kyrillischen Zeichen in ein Byte-Array " +
   "mit der Unicode-Codierung umwandeln");
Console.WriteLine();
Console.WriteLine("Originaler String: {0}", source);
byte[] result = System.Text.Encoding.Unicode.GetBytes(source);

// Byte-Array testweise ausgeben
Console.WriteLine("Das Byte-Array:");
for (int i = 0; i < result.Length; i++)
{
   Console.WriteLine("{0}: {1} ", result[i], (char)result[i]);
}

// String zurückkonvertieren
source = System.Text.Encoding.Unicode.GetString(result);
Console.WriteLine("Zurückkonvertierter String: {0}", source);
```

Listing 47: Umwandeln eines Strings in ein Byte-Array mit der Unicode-Codierung

Wenn Sie die Unicode-Codierung verwenden, werden pro Zeichen zwei Bytes gespeichert. Das erste Byte verwaltet den niedrigen und das zweite den hohen 8-Bit-Wert eines Zeichens. Bei Zeichen im 8-Bit-Bereich ist das zweite Byte natürlich immer mit 0 belegt. In Abbildung 31 sehen Sie aber, dass die kyrillischen Zeichen beide Bytes belegen.



Abbildung 31: Ein Byte-Array in der Unicode-Codierung (die kyrillischen Zeichen werden in der Konsole, die per Default kein Unicode kennt, als Fragezeichen dargestellt; die Zeichen-Ausgabe der kyrillischen Zeichen entspricht in der Abbildung den einzelnen Bytes dieser Zeichen)

39 Byte-Array in String lesen

Im Rezept 38 habe ich beschrieben, wie Sie einen String in ein Byte-Array umwandeln. Der Weg zurück ist natürlich auch möglich (und wird im genannten Rezept bereits eingesetzt ☺).

Dazu verwenden Sie die `GetString`-Methode eines zur im Byte-Array verwendeten Codierung passenden `Encoding`-Objekts. Grundlagen dazu habe ich bereits im genannten Rezept erläutert. Sie müssen beim Umwandeln lediglich beachten, dass Sie eine passende Codierung einsetzen.

Das Beispiel in Listing 48 erzeugt ein Unicode-Byte-Array und wandelt dies in einen String um. Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.Text` einbinden.

```
byte[] unicodeCharCodes = {0x41, 0, 0x42, 0, 0x20, 0, 0xE4, 0, 0xF6, 0};

// Byte-Array in String umwandeln
string result = Encoding.Unicode.GetString(unicodeCharCodes);

Console.WriteLine(result);
```

Listing 48: Umwandeln eines Byte-Arrays in einen String

Das Ergebnis dieses Programms ist der String "AB äö". Würden Sie statt des Unicode-`Encoding`-Objekts eines für die ISO-8859-1-Codierung verwenden, würde der String bei einer Ausgabe folgendermaßen aussehen: "A B ä ö". Die zusätzlichen Leerzeichen resultieren aus den 0-Bytes der Unicode-Daten, die bei einer Ausgabe als unsichtbares Zeichen erscheinen.

40 Auflistungen und Arrays sortieren

Das Sortieren von Auflistungen ist im .NET Framework 2.0 sehr einfach geworden, vorausgesetzt, Sie verwenden die generische `List`-Klasse aus dem Namensraum `System.Collections.Generic` oder eine `ArrayList`-Instanz. Diese besitzen nämlich eine Methode `Sort`, die laut der Dokumentation des .NET Framework 2.0 den besonders schnellen Quicksort-Algorithmus einsetzt. Für Arrays verwenden Sie die statische `Sort`-Methode der `Array`-Klasse, die ähnlich arbeitet.

Der Quicksort-Algorithmus geht die in zwei Hälften geteilte Liste durch und vergleicht jeweils zwei aufeinander folgende Objekte daraufhin, ob das eine größer ist als das andere. `Sort` ruft dazu für die einzelnen Objekte die Methode `CompareTo` aus der `IComparable`-Schnittstelle auf, sofern die in der Auflistung verwalteten Objekte diese Schnittstelle implementieren. Da dies für die Standard-Typen der Fall ist, können Sie Auflistungen dieser Typen problemlos sortieren:

```
// String-Auflistung erzeugen und füllen
List<string> personNames = new List<string>();
personNames.Add("zaphod");
personNames.Add("Trillian");
personNames.Add("arthur");
personNames.Add("ford");
personNames.Add("Marvin");
personNames.Add("Slartibartfaß");

// String-Auflistung per Voreinstellung sortieren
personNames.Sort();
```

Listing 49: Sortieren einer String-Auflistung

```
// Ergebnis ausgeben
foreach (string personName in personNames)
{
   Console.WriteLine(personName);
}
```

Listing 49: Sortieren einer String-Auflistung (Forts.)



Abbildung 32: Eine sortierte String-Auflistung

Wie Sie Abbildung 32 entnehmen können, werden Strings per Voreinstellung aufsteigend sortiert. Wenn Sie die Sortierung ändern wollen, können Sie am einzigen Argument der `Sort`-Methode einen Delegate vom Typ `Comparison<`*Typ*`>` übergeben. Ein solcher Delegate referenziert eine Methode, die zwei Instanzen vom angegebenen Typ übergeben bekommt und einen `int`-Wert zurückgibt. Die Rückgabe der Methode bestimmt mit einem Wert kleiner Null, dass das linke Objekt kleiner ist als das rechte. Null gibt an, dass beide Objekte gleich groß sind, und ein Wert größer null, dass das linke Objekt größer ist als das rechte.

Im folgenden Beispiel sorgt die Vergleichsmethode dafür, dass Strings absteigend sortiert werden:

```
/* Methode zum absteigenden Vergleich zweier Strings */
private static int StringComparer(string left, string right)
{
   return String.Compare(left, right) * -1;
}

...

// String-Auflistung über die Vergleichs-Methode sortieren
personNames.Sort(StringComparer);

// Ergebnis ausgeben
foreach (string personName in personNames)
{
   Console.WriteLine(personName);
}
```

Listing 50: Absteigende Sortierung einer String-Auflistung

Ein kleiner Hinweis am Rande: Die Übergabe des Methodennamens an `Sort` führt dazu, dass der Compiler eine Instanz des passenden Delegate erzeugt und übergibt. Der `Sort`-Aufruf im Beispiel entspricht also dem folgenden Aufruf:

```
personNames.Sort(new Comparison<string>(stringComparer));
```

Alternativ (und bei der im .NET Framework 2.0 eigentlich nicht mehr benötigten `ArrayList`-Klasse ausschließlich) zur `Comparison`-Instanz können Sie auch die Instanz einer Klasse übergeben, die die Schnittstelle `IComparer` implementiert. Diese Schnittstelle deklariert die Methode `Compare`, der wie beim `Comparison`-Delegate zwei Objekte übergeben werden. Diese Argumente werden auf dieselbe Weise miteinander verglichen wie bei der `Comparison`-Methode (allerdings müssen die übergebenen `object`-Referenzen dazu zunächst in den entsprechenden Typ konvertiert werden).



Abbildung 33: Eine absteigend sortierte String-Auflistung

Wollen Sie eigene Klassen problemlos sortierbar machen, können Sie in diesen die `IComparable`-Schnittstelle implementieren. `IComparable` besitzt lediglich die Methode `CompareTo`. Dieser Methode übergibt `Sort` das Objekt, das mit dem aktuellen verglichen werden soll. `CompareTo` muss die folgenden Werte zurückgeben, wenn Sie aufsteigend sortieren wollen:

- -1, wenn das aktuelle Objekt kleiner ist als das übergebene,
- 0, wenn beide Objekte gleich groß sind,
- 1, wenn das aktuelle Objekt größer ist als das übergebene.

`IComparable` liegt in einer normalen Version (mit einer `object`-Referenz) und in einer generischen Version vor. Sie sollten die generische Version einsetzen und als Typ Ihre Klasse angeben, wenn Sie lediglich Instanzen Ihrer Klasse miteinander vergleichen wollen (was ja wohl die Regel sein dürfte). Wollen Sie in einer (`ArrayList`-)Auflistung auch Instanzen anderer Klassen speichern und diese mit Ihrer Klasse vergleichen (was wohl nur selten Sinn macht), können Sie die normale Version von `IComparable` einsetzen. In diesem Fall sollten Sie aber darüber nachdenken, ob nicht der Einsatz einer Schnittstelle besser wäre, die beide Klassen implementieren (und die als Typ für `IComparable` angegeben wird).

Die Klasse `Person` in Listing 51 zeigt ein Beispiel für die Implementierung der `CompareTo`-Methode in der generischen Variante. In dieser Klasse vergleicht `CompareTo` den Nach- und den Vornamen der Person um zu ermitteln, welche Instanz größer ist.

```
public class Person : IComparable<Person>
{
   public string FirstName;
   public string LastName;

   public Person(string firstName, string lastName)
   {
      this.FirstName = firstName;
      this.LastName = lastName;
   }

   /* Implementierung der CompareTo-Methode */
   public int CompareTo(Person otherPerson)
   {
      // Vergleichen
      if (String.Compare(this.LastName,
          otherPerson.LastName, true) == 0 &&
          String.Compare(this.FirstName,
          otherPerson.FirstName, true) == 0)
      {
         // Beide Instanzen sind gleich
         return 0;
      }
      else if (String.Compare(this.LastName,
         otherPerson.LastName, true) < 0 ||
         (String.Compare(this.LastName,
         otherPerson.LastName, true) == 0 &&
         String.Compare(this.FirstName,
         otherPerson.FirstName, true) < 0))
      {
         // Diese Instanz ist kleiner als die andere
         return -1;
      }
      else
      {
         // Diese Instanz ist größer als die andere
         return 1;
      }
   }
}
```

Listing 51: Klasse, die die IComparable-Schnittstelle für ein korrektes Sortieren implementiert

Wenn Sie nun eine Auflistung von `Person`-Instanzen erzeugen und sortieren, erfolgt die Sortierung nach der `CompareTo`-Methode:

```
// Auflistung eigener Person-Objekte erzeugen
// und füllen
List<Person> persons = new List<Person>();
persons.Add(new Person("Zaphod", "Beeblebrox"));
persons.Add(new Person("Ford", "Prefect"));
persons.Add(new Person("Carl", "Dent"));
persons.Add(new Person("Arthur", "Miller"));
persons.Add(new Person("Tricia", "McMillan"));
persons.Add(new Person("Arthur", "Dent"));
persons.Add(new Person("Ford", "Puma"));

// Die Person-Auflistung sortieren und ausgeben
persons.Sort();
foreach (Person person in persons)
{
   Console.WriteLine(person.LastName + ", " + person.FirstName);
}
```

Listing 52: Sortieren einer Auflistung eigener Objekte



Abbildung 34: Sortieren eigener Objekte

41 Auflistungen und Arrays performant durchsuchen

In Auflistungen und Arrays können Sie sequenziell, mit `IndexOf` und mit `BinarySeach` suchen.

sequenzielle Suche

Die in der Regel schnellste und flexibelste Möglichkeit ist die eigene, sequenzielle Suche. Warum diese Suche normalerweise schneller als die anderen ist, kläre ich nach der Beschreibung von `IndexOf` und am Ende der Beschreibung von `BinarySearch`. Flexibel ist diese Art der Suche aus dem Grund, dass Sie selbst entscheiden können, welche Felder oder Eigenschaften der Objekte oder ob Sie die Referenzen vergleichen wollen.

Die folgende Struktur implementiert z. B. Kreisdaten:

```
public struct CircleStruct
{
   public int X;
```

Listing 53: Struktur zur Speicherung von Kreisdaten

```
   public int Y;
   public int Radius;

   public CircleStruct(int x, int y, int radius)
   {
      this.X = x;
      this.Y = y;
      this.Radius = radius;
   }
}
```

Listing 53: Struktur zur Speicherung von Kreisdaten (Forts.)

Listing 54 zeigt, wie Sie in einer Auflistung von Instanzen der `CircleStruct`-Struktur nach Kreisen mit einem Radius von 50 suchen:

```
// Auflistung von 1.000.000 Circle-Werttyp-Objekten erzeugen
List<CircleStruct> circleValueTypeList = new List<CircleStruct>();
Random random = new Random();
for (int i = 0; i < 1000000; i++)
{
   circleValueTypeList.Add(new CircleStruct(random.Next(1, 11),
      random.Next(1, 11), random.Next(1, 101)));
}

// Alle Kreise mit dem Radius 50 suchen
List<int> result = new List<int>();
for (int i = 0; i < circleValueTypeList.Count; i++)
{
   if (circleValueTypeList[i].Radius == 50)
   {
      result.Add(i);
   }
}
```

Listing 54: Eigene sequenzielle Suche

Die sequenzielle Suche funktioniert natürlich genauso mit einer Auflistung oder einem Array von Referenztypen.

Suche mit IndexOf

Auflistungen besitzen normalerweise eine `IndexOf`-Methode, die den Index des Objekts zurückgibt, das Sie am ersten Argument übergeben. Zur Suche in Arrays können Sie die statische `IndexOf`-Methode der `Array`-Klasse verwenden.

Diese Methode sucht sequenziell nach dem übergebenen Objekt. Die Objekte werden dabei intern über deren `Equals`-Methode verglichen. Ist diese nicht in den Objekten überschrieben, werden bei Referenztypen die Referenzen und bei Werttypen alle öffentlichen Felder und Eigenschaften verglichen. Speichern Sie in einer Auflistung z. B. Instanzen einer `Circle`-Klasse und übergeben Sie `IndexOf` eine Referenz auf ein `Circle`-Objekt, sucht diese Methode nach genau diesem Objekt. Andere Objekte, die denselben Inhalt aufweisen wie das Suchobjekt (in unserem Beispiel dieselben Werte für `X`, `Y` und `Radius`), werden nicht gefunden. Speichern Sie in der Auflistung Instanzen einer Struktur, wird natürlich nicht nach der Referenz gesucht. In diesem Fall findet `IndexOf` alle Objekte, deren Inhalt dem des Suchobjekts entspricht. Beim Suchen nach einem Kreis mit `X` = 5, `Y` = 10 und `Radius` = 100 werden alle Kreise gefunden, deren Eigenschaften dieselben Werte aufweisen. Implementieren die Objekte allerdings die von `object` geerbte `Equals`-Methode, entspricht das Suchergebnis natürlich dieser Implementierung.

Um alle Objekte zu finden, müssen Sie `IndexOf` in einer Schleife aufrufen.

Listing 55 zeigt, wie Sie mit `IndexOf` in einer Auflistung von Strukturen nach Objekten suchen:

```
// Auflistung von 1.000.000 Circle-Werttyp-Objekten erzeugen
List<CircleStruct> circleValueTypeList = new List<CircleStruct>();
Random random = new Random();
for (int i = 0; i < 1000000; i++)
{
   circleValueTypeList.Add(new CircleStruct(random.Next(1, 11),
      random.Next(1, 11), random.Next(1, 101)));
}

// Mit IndexOf in der Auflistung der Werttyp-Kreise suchen.
// Bei der Suche in einer Auflistung von Werttypen werden immer die
// öffentlichen Eigenschaften und Felder der Objekte verglichen.
CircleStruct compareCircle = new CircleStruct(5, 10, 50);
List<int> result = new List<int>();

// Ersten Kreis suchen
int index = circleValueTypeList.IndexOf(compareCircle);
while (index > -1)
{
   result.Add(index);

   // Weitersuchen
   index = circleValueTypeList.IndexOf(compareCircle, index + 1);
}
```

Listing 55: IndexOf-Suche nach Werttypen in einer Auflistung

Die Performance

Die Performance der sequenziellen Suche ist nach meinen Performance-Messungen (mit einem `Array` und einer `List`-Auflistung) wesentlich besser als die Suche über `IndexOf`. Das sequenzielle Suchen nach Kreisen mit `X` = 5, `Y` = 10 und `Radius` = 50 in einer Auflistung mit 1.000.000

Circle-Werttyp-Objekten benötigte z. B. lediglich ca. 0,018 Sekunden, das Suchen mit IndexOf hingegen ca. 0,13 Sekunden. Die Suche mit IndexOf nach einem bestimmten Objekt in einer Auflistung mit 1.000.000 Circle-Referenztyp-Objekten benötigte ca. 0,022 Sekunden, die sequenzielle Suche nach demselben Objekt hingegen lediglich ca. 0,006 Sekunden. IndexOf ist also für die performante Suche nicht geeignet.

Ich habe auch eine Erklärung für die bessere Performance beim eigenen sequenziellen Suchen: IndexOf verwendet intern eine Schleife über alle Elemente. Innerhalb der Schleife wird das zu suchende Objekt über die Equals-Methode verglichen. Der Aufruf einer Methode benötigt natürlich ein wenig mehr Zeit als ein direkter Vergleich von Objekten.

IndexOf ist also eigentlich nur dann für eine Suche geeignet, wenn Sie nach einem Objekt suchen, das nur einmal in der Liste vorkommt, und Sie den Vergleich über Equals verwenden wollen.

Hinweis

In diesem Fall ist der Performanceverlust minimal, der Codierungsaufwand dafür aber geringer als bei der sequenziellen Suche.

Suchen mit BinarySearch

Die dritte Suchmöglichkeit ist die Suche mit der BinarySearch-Methode, die in der Array-Klasse (als statische Methode) und in vielen Auflistungen zur Verfügung steht. BinarySearch sucht nach dem übergebenen Objekt und gibt den Index des gefundenen Elements zurück. Wird kein Element gefunden, das zum Suchobjekt passt, gibt diese Methode -1 zurück.

BinarySearch führt eine effiziente binäre Suche in den Elementen aus, was allerdings voraussetzt, dass die Liste aufsteigend sortiert ist. Ohne eine aufsteigende Sortierung führt BinarySearch zu keinem oder einem falschen Ergebnis! Das Sortieren setzt voraus, dass der in der Auflistung verwaltete Typ die CompareTo-Methode der IComparable-Schnittstelle implementiert.

Das notwendige Sortieren ist der Knackpunkt dieser Methode, da es sehr viel Zeit in Anspruch nimmt. BinarySearch lohnt sich nur dann, wenn Sie mehrfach in demselben Array bzw. in derselben Auflistung suchen müssen oder wenn diese sowieso schon sortiert sind.

Listing 56 zeigt eine erweiterte CircleClass-Klasse, die das Sortieren und Suchen nach dem Radius ermöglicht.

```
public class CircleClass: IComparable<Circle>
{
   public int x;
   public int y;
   public int Radius;

   public CircleClass (int x, int y, int radius)
```

Listing 56: Klasse zur Speicherung von vergleichbaren Kreisdaten

```
   {
      this.x = x;
      this.y = y;
      this.Radius = radius;
   }

   /* Implementierung der CompareTo-Methode */
   public int CompareTo(CircleClass otherCircle)
   {
      return (this.Radius.CompareTo(otherCircle.Radius));
   }
}
```

Listing 56: Klasse zur Speicherung von vergleichbaren Kreisdaten (Forts.)

Die `CompareTo`-Methode ist in diesem Beispiel sehr einfach, weil zum Vergleich die gleichnamige Methode des `Radius`-Felds aufgerufen werden kann.

Um mit `BinarySearch` zu suchen, rufen Sie diese Methode zunächst einmal auf, um das erste Objekt zu finden. Dieses Objekt ist aber nicht unbedingt auch das erste der Objekte in der Auflistung, die die Suchkriterien erfüllen. Das liegt an der Natur des Suchalgorithmus, der die Liste immer wieder in Hälften aufteilt und das Objekt in der Mitte mit dem Suchobjekt vergleicht. Das erste gefundene Objekt kann also irgendwo innerhalb der Unter-Liste der Objekte liegen, die die Suchkriterien erfüllen.

Um alle Objekte zu finden, müssen Sie nach der ersten Suche die Liste ab dem gefundenen Index einmal nach oben und einmal nach unten durchgehen. Ich zeige dies am Beispiel der bereits oben verwendeten Auflistung von 1.000.000 Kreis-Objekten:

```
// Auflistung von 1.000.000 Kreis-Referenztyp-Objekten erzeugen
List<CircleClass> circleReferenceTypeList = new List<CircleClass>();
Random random = new Random();
for (int i = 0; i < 1000000; i++)
{
   circleReferenceTypeList.Add(new CircleClass(random.Next(1, 11),
      random.Next(1, 11), random.Next(1, 101)));
}
// Ersten Kreis mit einem Radius von 50 suchen
CircleClass compareCircle = new CircleClass(0, 0, 50);
List<int> result = new List<int>();
int index = circleReferenceTypeList.BinarySearch(compareCircle);
if (index > -1)
{
   // Ergebnis speichern
   result.Add(index);
}
```

Listing 57: Suchen mit BinarySearch

```
// Nächsten Kreis nach unten suchen
for (int i = index + 1; i < circleReferenceTypeList.Count; i++)
{
   if (circleReferenceTypeList[i].CompareTo(compareCircle) == 0)
   {
      // Ergebnis speichern
      result.Add(i);
   }
   else
   {
      // Schleife abbrechen
      break;
   }
}

// Nächsten Kreis nach oben suchen
for (int i = index - 1; i > -1; i--)
{
   if (circleReferenceTypeList[i].CompareTo(compareCircle) == 0)
   {
      // Ergebnis speichern
      result.Add(i);
   }
   else
   {
      // Schleife abbrechen
      break;
   }
}
```

Listing 57: Suchen mit BinarySearch (Forts.)

Hinweis

Wie bereits gesagt ist die Performance von `BinarySearch` zwar gut, das notwendige Sortieren benötigt aber sehr viel Zeit. In meinem Performancetest benötigte das Sortieren 0,94 Sekunden. Die Suche nach allen Kreisen mit dem Radius 50 benötigte dann allerdings nur noch 0,0013 Sekunden. Eine sequenzielle Suche benötigte die mehr als 10-fache Zeit, nämlich 0,015 Sekunden. Das Suchen mit `BinarySearch` ist also prinzipiell nur dann ein echter Performancegewinn, wenn das Array bzw. die Auflistung bereits (aufsteigend) sortiert ist. Sind die Daten nicht sortiert, ist ein Performancegewinn erst dann spürbar, wenn die Suche recht häufig ausgeführt werden muss. In meinem Test wäre das etwa 80-mal. Der Gewinn hängt aber natürlich auch noch davon ab, welche Objekte gesucht werden. Ich denke, Sie sollten das bei Performance-kritischen Anwendungen im Einzelfall prüfen.

42 Typisierte Auflistungen erzeugen

Dieses Rezept, das in der ersten Version des Codebook sehr wichtig war, ist nun sehr einfach geworden. Denn das .NET Framework stellt in den Namensräumen `System.Collections.Generic` und `System.Collections.ObjectModel` einige generische Klassen für verschiedene typisierte Listen zur Verfügung. Etwas verwirrend ist zum einen, dass technisch prinzipiell identische Auflistungen doppelt vorkommen (z. B. die Collections `System.Collections.Generic.List` und `System.Collections.ObjectModel.Collection`) und zum anderen, dass diese in zwei Namensräumen vorkommen. Das liegt daran, dass der Namensraum `System.Collections.ObjectModel` ursprünglich für den OR-Mapper vorgesehen war, den Microsoft der Version 2.0 des .NET Framework mitgeben wollte. Die dort verwendeten speziellen Auflistungsklassen hat Microsoft einfach im Framework belassen, wahrscheinlich weil diese in einigen Fällen sehr praktisch sind. Warum diese allerdings nicht in den `System.Collections.Generic` verschoben wurden, ist mir nicht ganz klar.

Zur Übersicht folgen hier erst zwei Tabellen, die die einzelnen generischen Auflistungsklassen beschreiben.

Klasse	Beschreibung
`Dictionary <TKey, TValue>`	Implementiert ein Dictionary (bzw. eine Hashtable), bei der jedes gespeicherte Objekt mit einem Schlüssel assoziiert ist. Der Typ des Schlüssels wird in `TKey` angegeben, der Typ des gespeicherten Werts in `TValue`. Elemente fügen Sie über die `Add`-Methode hinzu. Auf die Elemente eines Dictionarys können Sie über den Indexer zugreifen, indem Sie den Schlüssel angeben. Die `ContainsKey`-Methode ermöglicht die Prüfung darauf, ob ein Element mit einem bestimmten Schlüssel in der Liste gespeichert ist. Über die `Values`-Eigenschaft erhalten Sie eine Auflistung der gespeicherten Werte, über die Sie diese auch durchgehen können. Die Reihenfolge dieser Auflistung entspricht allerdings nicht unbedingt der Reihenfolge, in der Sie die Objekte hinzugefügt haben.
`HashSet<T>`	Ein `HashSet` ist eine unsortierte Auflistung (`Collection`), die zusätzlich Standard-Mengenoperationen wie `IntersectWith` (Bilden der Schnittmenge), `UnionWith` (Bilden der gemeinsamen Menge), `IsSubsetOf` (gibt an, ob die Objekte im `HashSet` eine Untermenge der Objekte in der übergebenen Auflistung sind) und `IsSupersetOf` (gibt an, ob die Objekte im `HashSet` eine Übermenge der Objekte in der übergebenen Auflistung sind) zur Verfügung stellt.
`LinkedList<T>`	Implementiert eine Liste, bei der einzelne Elemente mit ihrem Vorgänger und ihrem Nachfolger verknüpft sind. Über verschiedene Add-Methoden können Sie Elemente an bestimmten Positionen einfügen oder über Remove-Methoden entfernen. Über die `First`-Eigenschaft erreichen Sie den ersten Knoten, über die `Last`-Eigenschaft den letzten. Jeder Knoten kennt über seine `Previous`-Eigenschaft seinen vorhergehenden und über die `Next`-Eigenschaft seinen nachfolgenden Knoten. Beim ersten Knoten ist `Previous` natürlich `null`, beim letzten Knoten speichert `Next` `null`. Über die Eigenschaft `Value` eines Knotens erreichen Sie den im Knoten gespeicherten Wert.
`List<T>`	`List` implementiert eine Liste von Objekten, die über einen Integer-Index angesprochen werden können (Collection). Objekte fügen Sie über die `Add`-Methode hinzu. Über den Indexer können Sie einzelne Elemente auslesen, indem Sie deren Index angeben. Eine `List`-Auflistung unterstützt natürlich auch das Durchgehen mit `foreach`.

Tabelle 6: Die generischen Auflistungsklassen aus dem Namensraum System.Collections.Generic

Klasse	Beschreibung
`Queue<T>`	Ein `Queue` stellt eine FIFO-Auflistung (First In First Out) zur Verfügung. Über die `Enqueue`-Methode können Sie Elemente hinzufügen, über die `Dequeue`-Methode auslesen und entfernen. `Peek` ermöglicht das Auslesen eines Objekts ohne dieses zu entfernen. Über die Eigenschaft `Count` erhalten Sie (wie üblich) die Anzahl der gespeicherten Elemente.
`SortedDictionary<TKey, TValue>`	Implementiert ein Dictionary, dessen Elemente automatisch nach dem Schlüssel sortiert werden.
`SortedList<TKey, TValue>`	Implementiert ein Dictionary (!), dessen Elemente automatisch nach dem Schlüssel sortiert werden. Der Unterschied zwischen `SortedDictionary` und `SortedList` ist: ▶ dass `SortedList` weniger Speicher benötigt als `SortedDictionary`, ▶ dass `SortedDictionary` unsortierte Daten schneller hinzufügt als `SortedList`, ▶ dass `SortedList` schneller ist, wenn die Liste in einem Durchgang mit sortierten Daten gefüllt wird
`Stack<T>`	Implementiert eine LIFO-Auflistung (Last in First Out). Über die `Push`-Methode können Sie Elemente hinzufügen, über die `Pop`-Methode auslesen und entfernen. `Peek` ermöglicht das Auslesen eines Objekts ohne dieses zu entfernen. Über die Eigenschaft `Count` erhalten Sie die Anzahl der gespeicherten Elemente.

Tabelle 6: Die generischen Auflistungsklassen aus dem Namensraum System.Collections.Generic (Forts.)

Klasse	Beschreibung
`Collection<T>`	Implementiert eine Liste von Objekten, die über einen Integer-Index referenziert werden können, und ist deswegen mit der `List`-Klasse vergleichbar. `Collection` bietet jedoch zum einen weniger Möglichkeiten (z. B. nicht die binäre Suche) und zum anderen einige geschützte Methoden, über die Sie beim Ableiten einer Klasse das Verhalten der Auflistung beim Hinzufügen, Verändern oder Löschen von Elementen beeinflussen können. `Collection` ist aus diesem Grunde ideal als Basisklasse für Auflistungen geeignet, die von Methoden Ihrer Klassen zurückgegeben werden.
`KeyedCollection<TKey, TItem>`	Implementiert eine Auflistung, deren Elemente mit einem Schlüssel assoziiert gespeichert werden (ähnlich einem `Dictionary`. Auf die gespeicherten Elemente können Sie allerdings über den Integer-Index und über den Schlüssel zugreifen. `KeyedCollection` bietet ähnlich `Collection` einige geschützte Methoden, über die Sie in abgeleiteten Klassen das Verhalten beim Hinzufügen, Ändern und Löschen von Elementen beeinflussen können. Diese Klasse wird in Rezept 43 beschrieben.
`ReadOnlyCollection<T>`	Eine `ReadOnlyCollection` ist eine Auflistung, die allerdings nur einen schreibgeschützten Zugriff erlaubt. Damit ist gemeint, dass keine Elemente hinzugefügt, gelöscht oder neu zugewiesen werden können. Das Verändern der Eigenschaften von gespeicherten Referenztypen (nicht von Werttypen) ist allerdings möglich, da die Liste in diesem Fall ja lediglich Referenzen auf die Objekte verwaltet. Zum Erzeugen einer `ReadOnlyCollection` benötigen Sie eine andere Auflistung, die die `IList`-Schnittstelle implementiert. Diese übergeben Sie dem Konstruktor. Eine `ReadOnlyCollection`-Auflistung eignet sich hervorragend für die schreibgeschützte Rückgabe von Auflistungen in Methoden eigener Klassen.

Tabelle 7: Die generischen Auflistungsklassen aus dem Namensraum System.Collections.ObjectModel

Das folgende Beispiel zeigt, wie Sie verschiedene Listen der Klasse `Person` (aus dem Rezept 40) erzeugen. Das Beispiel benötigt den Import der Namensräume `System.Collections.Generic` und `System.Collections.ObjectModel`.

```
// List-Instanz mit Person-Objekten erzeugen und füllen
List<Person> personList = new List<Person>() {
   new Person("Tricia", "McMillan"),
   new Person("Zaphod", "Beeblebrox"),
   new Person("Arthur", "Dent"),
   new Person("Ford", "Prefect")};

// List-Instanz sortieren
personList.Sort();

// List-Instanz durchgehen
Console.WriteLine("Sortierte List-Instanz:");
foreach (Person p in personList)
{
   Console.WriteLine(p.FirstName + " " + p.LastName);
}

// Dictionary erzeugen und füllen
Dictionary<int, Person> personDictionary =
   new Dictionary<int, Person>();
personDictionary.Add(1000, new Person("Tricia", "McMillan"));
personDictionary.Add(1001, new Person("Zaphod", "Beeblebrox"));
personDictionary.Add(1002, new Person("Arthur", "Dent"));
personDictionary.Add(1003, new Person("Ford", "Prefect"));

// Person suchen
Console.WriteLine();
Console.WriteLine("Suchen im Dictionary");
int personId = 1001;
if (personDictionary.ContainsKey(personId))
{
   Console.WriteLine("Gefunden: " +
      personDictionary[personId].FirstName + " " +
      personDictionary[personId].LastName);
}
else
{
   Console.WriteLine("Nicht gefunden");
}
```

Listing 58: Verwenden verschiedener generischer Listen-Klassen

```
// Personen durchgehen
Console.WriteLine();
Console.WriteLine("Elemente des Dictionary durchgehen");
foreach (Person p in personDictionary.Values)
{
   Console.WriteLine(p.FirstName + " " + p.LastName);
}

// Collection erzeugen
Console.WriteLine();
Collection<Person> personCollection = new Collection<Person>() {
   new Person("Tricia", "McMillan"),
   new Person("Zaphod", "Beeblebrox"),
   new Person("Arthur", "Dent"),
   new Person("Ford", "Prefect")};

// Collection durchgehen
Console.WriteLine("Collection:");
foreach (Person p in personCollection)
{
   Console.WriteLine(p.FirstName + " " + p.LastName);
}

// Stack erzeugen und füllen
Console.WriteLine();
Console.WriteLine("Objekte in einem Stack verwalten");
Stack<Person> personStack = new Stack<Person>();
personStack.Push(new Person("Tricia", "McMillan"));
personStack.Push(new Person("Zaphod", "Beeblebrox"));
personStack.Push(new Person("Arthur", "Dent"));
personStack.Push(new Person("Ford", "Prefect"));

// Letzte Person holen
Person person = personStack.Pop();
Console.WriteLine("Letzte Person im Stack war: " +
   person.FirstName + " " + person.LastName);

// ReadOnlyCollection erzeugen
Console.WriteLine();
Console.WriteLine("Objekte in einer ReadOnlyCollection verwalten");
List<Person> baseList = new List<Person> {
   new Person("Tricia", "McMillan"),
   new Person("Zaphod", "Beeblebrox"),
   new Person("Arthur", "Dent"),
   new Person("Ford", "Prefect")};
```

Listing 58: Verwenden verschiedener generischer Listen-Klassen (Forts.)

```
ReadOnlyCollection<Person> readOnlyCollection =
   new ReadOnlyCollection<Person>(baseList);
foreach (Person p in readOnlyCollection)
{
   Console.WriteLine(p.FirstName + " " + p.LastName);
}

// readOnlyCollection[0] = new Person("Ford", "Prefect");
// Eine Zuweisung ist nicht möglich
// Allerdings kann das Objekt verändert werden, da es sich um einen
// Referenztypen handelt
readOnlyCollection[0].FirstName = "Ford";
readOnlyCollection[0].LastName = "Prefect";

// HashSet erzeugen
Console.WriteLine();
Console.WriteLine("Mit HashSets arbeiten");
HashSet<int> numberHashSet1 = new HashSet<int> {1, 2, 3, 4, 5, 6};
HashSet<int> numberHashSet2 = new HashSet<int> {3, 4, 5};
HashSet<int> numberHashSet3 = new HashSet<int> {1, 2, 4, 6, 7};
HashSet<int> numberHashSet4 = new HashSet<int> { 1, 3, 5, 7, 9, 11};

// Überprüfen, ob das HashSet 1 eine Übermenge von HashSet 2 ist
// Anm.: Um lesbare Ausgaben zu erzielen, habe ich im Beispiel die
// Erweiterungsmethode ToStringList implementiert, die die Zahlen als
// sortierte String-Liste zurückgibt.
if (numberHashSet1.IsSupersetOf(numberHashSet2))
{
   Console.WriteLine(numberHashSet1.ToStringList() + " ist eine Übermenge von " +
      numberHashSet2.ToStringList());
}

// Überprüfen, ob das HashSet 2 eine Untermenge von HashSet 1 ist
if (numberHashSet2.IsSubsetOf(numberHashSet1))
{
   Console.WriteLine(numberHashSet2.ToStringList() + " ist eine Untermenge von " +
      numberHashSet1.ToStringList());
}

// Ein HashSet mit einem anderen zusammenführen
Console.WriteLine();
Console.WriteLine("Zwei HashSets zusammenführen:");
Console.Write(numberHashSet3.ToStringList() + " mit " +
   numberHashSet2.ToStringList()  +  " = ");
numberHashSet3.UnionWith(numberHashSet2);
Console.WriteLine(numberHashSet3.ToStringList());
```

Listing 58: Verwenden verschiedener generischer Listen-Klassen (Forts.)

```
Console.WriteLine();
Console.WriteLine("Schnittmenge aus zwei HashSets:");
Console.Write(numberHashSet4.ToStringList() + " mit " +
   numberHashSet1.ToStringList() + " = ");
numberHashSet4.IntersectWith(numberHashSet1);
Console.WriteLine(numberHashSet4.ToStringList());
```

Listing 58: Verwenden verschiedener generischer Listen-Klassen (Forts.)

43 Schnelle Auflistung mit Schlüssel- und Indexzugriff

Die Klasse `KeyedCollection` aus dem Namensraum `System.Collections.ObjectModel` erlaubt die Erzeugung einer Auflistung, auf deren Elemente Sie über einen Schlüssel oder einen Integer-Index zugreifen können. Diese Flexibilität wird wahrscheinlich mit etwas mehr Speicherverbrauch als bei einem `Dictionary` erkauft, dafür ist der Zugriff über den Schlüssel aber sogar etwas schneller (laut eigener Performancemessung).

Und mit `KeyedCollection` arbeiten zu können, müssen Sie von der abstrakten Klasse eine neue Klasse ableiten, die die Methode `GetKeyForItem` implementiert. Diese Methode liefert den Schlüssel für ein gespeichertes Objekt. Die Rückgabe der Methode muss dem Typ entsprechen, der bei der Deklaration der `KeyedCollection` als Schlüsseltyp angegeben wurde.

Das folgende Beispiel soll `User`-Objekte verwalten, die folgendermaßen deklariert sind:

```
/* Beispiel-Klasse für die Speicherung von Objekten
   in einer KeyedCollection */
public class User
{
   public string LoginName;
   public string FirstName;
   public string LastName;
   public User(string loginName, string firstName, string lastName)
   {
      this.LoginName = loginName;
      this.FirstName = firstName;
      this.LastName = lastName;
   }
}
```

Listing 59: Beispiel-Klasse zur Speicherung in einer Auflistung

Listing 60 zeigt die Deklaration einer von `KeyedCollection` abgeleiteten Klasse zur Speicherung von User-Objekten.

```
public class Users : KeyedCollection<string, User>
{
   /* Implementierung der GetKeyForItem-Methode */
   protected override string GetKeyForItem(User user)
   {
      return user.LoginName;
   }
}
```

Listing 60: Beispiel-Auflistung mit Zugriff über einen Schlüssel oder einen Integer-Index

Bei der Anwendung der Auflistung können Sie nun über einen Integer-Index auf das Objekt an einer bestimmten Position oder über den Schlüssel auf ein bestimmtes Objekt zugreifen:

```
// Auflistung erzeugen
Users users = new Users();

// Objekte anhängen
users.Add(new User("zaphod", "Zaphod", "Beeblebrox"));
users.Add(new User("ford", "Ford", "Prefect"));
users.Add(new User("tricia", "Tricia", "McMillan"));
users.Add(new User("arthur", "Arthur", "Dent"));

// Alle Objekte über den Integer-Index durchgehen
for (int i = 0; i < users.Count; i++)
{
   Console.WriteLine(users[i].FirstName + " " + users[i].LastName);
}
// Ein Objekt nach seinem Schlüssel ermitteln
User user = users["tricia"];
Console.WriteLine(user.FirstName + " " + user.LastName);
```

Listing 61: Beispiel-Anwendung der von KeyedCollection abgeleiteten Klasse

Hinweis

Falls Sie für den Schlüssel einen `int`-Wert verwenden, können Sie nur über den Schlüssel auf die gespeicherten Objekte zugreifen. Ein Zugriff über den Integer-Index ist (logischerweise) dann nicht möglich.

44 Das Programm für eine definierte Zeit anhalten

Manchmal ist es sinnvoll, ein Programm für eine kurze Zeit anzuhalten. Wenn Sie z. B. in einer Schleife eine Berechnung ausführen und das jeweilige Ergebnis in einem `Label` auf einem Windows-Formular ausgeben wollen, ist die Berechnung meist so schnell, dass Windows mit der Aktualisierung des Labels nicht nachkommt. Wenn Sie allerdings eine kleine Pause einbauen, erfolgt die Ausgabe in den durch die Pause bestimmten Abständen. Eine andere (wenn auch nicht allzu elegante) Einsatzmöglichkeit ist das Pausieren eines Info- oder

Splash-Formulars beim Start der Anwendung, damit der Anwender genügend »Zeit« hat, die dort ausgegebenen Informationen zu lesen.

Eine definierte Pause können Sie einfach über die statische `Sleep`-Methode der `Thread`-Klasse aus dem Namensraum `System.Threading` erzeugen. Dazu übergeben Sie ein `TimeSpan`-Objekt oder einfach einen `int`-Wert mit der Angabe der Pausenzeit in Millisekunden:

```
Thread.Sleep(1000);
```

`Sleep` hält den aktuellen Thread an (also den Thread, in dem die aktuelle Methode gerade ausgeführt wird). Wenn Sie keine speziellen Threads einsetzen, werden Ihre Methoden im Hauptthread der Anwendung ausgeführt. `Sleep` hält dann also die gesamte Anwendung an. Dieses Pausieren bewirkt zwar, dass der Anwender mit Windows problemlos weiterarbeiten kann, das Programm kann aber innerhalb der Pause nicht weiter bedient werden. Sogar das Verschieben des Fensters ist innerhalb der Pause nicht möglich. Für viele Anwendungen ist dieser Effekt eigentlich nicht geeignet, da dem Anwender das Verschieben des Formulars und oft auch ein Bedienen erlaubt sein sollten. Um dieses Problem zu lösen, müssen Sie einen separaten Thread für die eigentliche Aufgabe einsetzen, was ich hier aber nicht weiter behandle.

45 Die Nachrichten einer Exception und ihrer inneren Exceptions ermitteln

In den meisten Fällen ist die Nachricht einer Exception, die Sie aus der Eigenschaft `Message` des `Exception`-Objekts auslesen, sehr aussagekräftig. In diesen Fällen benötigen Sie dieses Rezept wohl nicht ☺. In einigen Fällen versteckt sich der eigentliche Fehler aber leider in einer der inneren Exceptions, die Sie über die Eigenschaft `InnerException` erreichen. Die Nachricht der Exception der ersten Ebene ist in diesen Fällen zumeist lediglich sehr allgemein gehalten, manchmal aber leider auch vollkommen irreführend. Dies ist besonders dann der Fall, wenn .NET-Methoden auf COM-Objekte zurückgreifen.

Ein Beispiel für dieses Problem ist das Senden von E-Mails, wie ich es in Rezept 211 beschreibe. Wird das Senden über einen SMTP-Server ausgeführt, und dieser läuft gerade nicht, resultiert dies in einer Exception mit der Nachricht »Fehler beim Senden von Mail«. Die erste innere Exception meldet »Die Verbindung mit dem Remoteserver kann nicht hergestellt werden«, die zweite (und letzte) »Es konnte keine Verbindung hergestellt werden, da der Zielcomputer die Verbindung verweigerte«.

Da dieses Problem meist Exceptions betrifft, die im Programm nicht erwartet werden, sollten Sie auf jeden Fall die Nachrichten aller Ebenen ausgeben oder zumindest protokollieren. Nur so ist sichergestellt, dass Sie Fehler leichter eingrenzen können, die während der Anwendung Ihres Programms auftreten.

Die Methode `GetExceptionMessages` in Listing 62 liest alle Nachrichten der übergebenen und der inneren Exceptions aus und gibt diese in einem String getrennt durch Zeilenvorschübe zurück. `GetExceptionMessages` ruft sich für alle inneren Exceptions rekursiv selbst auf. Da es vorkommt, dass die Meldung der äußeren und/oder der inneren Exceptions gleich sind, überprüft die Methode, ob die ermittelten Meldungen eindeutig sind, und verwirft die Duplikate.

`SoapExceptions`, die dann auftreten, wenn aufgerufene Webdienst-Methoden Exceptions werfen, erfahren eine Sonderbehandlung. Bei diesen Exceptions existiert nach meinen Erfahrun-

gen und Informationen aus dem Internet keine innere Exception. Die Nachricht der SoapException ist leider (zumindest für den Endanwender) recht kryptisch. Eine solche Nachricht sieht prinzipiell (zumindest für .NET-Webdienste) so aus:

```
Die Anforderung konnte vom Server nicht verarbeitet werden. ---> <Nachricht der
eigentlichen Ausnahme>
```

Dies gilt für den Fall, dass in der Webanwendung der Modus der CustomErrors-Einstellung auf on geschaltet ist (oder auf remoteOnly und der Aufruf der Webmethode stammt von einem anderen Rechner). Ist der Modus der CustomErrors-Einstellung auf off geschaltet (oder auf remoteOnly und der Aufruf stammt von demselben Rechner), ist die Nachricht detaillierter:

```
System.Web.Services.Protocols.SoapException: Die Anforderung konnte vom Server nicht
verarbeitet werden. ---> <Typ der eigentlichen Exception>: <Nachricht der eigentlichen
Exception>

<Stack-Trace>

 --- Ende der internen Ausnahmestapelüberwachung ---
```

Die Auswertung der eigentlichen Nachricht gestaltet sich damit leider etwas kompliziert.

Hinweis

Die SoapException-Klasse besitzt auch noch die Eigenschaft Detail, die für Detailinformationen zur aufgetretenen Exception vorgesehen ist. Diese Eigenschaft, die ein XmlNode-Objekt referenziert, muss allerdings vom Erzeuger der SoapException explizit gefüllt werden. Leider wird Detail nicht gefüllt, wenn die Webdienstmethode lediglich eine »normale« Ausnahme wirft oder weitergibt (die dann automatisch in eine SoapException umgewandelt wird). Eine Webdienstmethode sollte eigentlich alle auftretenden Exceptions abfangen und explizit eine SoapException werfen, der die Detailinformationen in Detail mitgegeben werden. Dummerweise handelt es sich dabei aber um ein nicht standardisiertes XML-Format: Der Entwickler der Webdienstmethode kann beliebige XML-Elemente an den Detail-Wurzel-Knoten anhängen. Die Auswertung wird dadurch leider sehr schwierig und kann nicht generalisiert werden. Deswegen beschränkt sich GetExceptionMessages darauf, die Nachricht der SoapException auszuwerten.

Der Start der Nachricht wird an dem String "--->" gefolgt von beliebigem Text und einem Doppelpunkt erkannt. Das Ende der Nachricht ist etwas schwerer zu erkennen, da es ja sein kann, dass der Stack-Trace an die Nachricht angehängt ist. Der Start des Stack-Trace könnte daran erkannt werden, dass die Zeile mit " bei" beginnt, aber das wäre sehr unsicher und müsste für alle möglichen Sprachen implementiert werden. Deshalb werte ich das Muster "\n " als Anfang des Stack-Trace.

Da ich in Webanwendungsprojekten immer wieder Probleme mit auftretenden Exceptions vom Typ HttpException und der davon abgeleiteten HttpUnhandledException[3] (die in ihrer Nachricht nur eine allgemeine Fehlermeldung enthalten) hatte, habe ich die Behandlung solcher Ausnahmen auch noch in die GetExceptionMessages-Methode integriert.

3. Eine HttpUnhandledException tritt z. B. auf, wenn eine Exception nicht abgefangen, und erst im Error-Ereignis der Application (in der *Global.asax*) ausgewertet wird.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Text.RegularExpressions` importieren. Wenn Sie den Teil mit der Auswertung der `HttpException`-Ausnahmen übernehmen, müssen Sie außerdem die Assembly *System.Web* referenzieren.

```
public static string GetExceptionMessages(Exception ex)
{
   string messages = null;

   if (ex is System.Web.HttpException)
   {
      // Bei einer HttpException sollte GetBaseException aufgerufen
      // werden um den zugrunde liegenden Fehler herauszufinden
      ex = ((System.Web.HttpException)ex).GetBaseException();
   }

   // SoapExceptions müssen separat behandelt werden, da diese
   // einen allgemeinen Text, die Nachricht der ursprünglichen
   // Exception und den StackTrace in der Message-Eigenschaft,
   // und leider keine InnerException verwalten.
   if (ex is System.Web.Services.Protocols.SoapException)
   {
      // Nachricht auslesen
      messages = ex.Message;

      // Löschen des einleitenden Textes, der mit ---> beginnt
      int pos1 = messages.IndexOf("--->");
      if (pos1 > -1)
      {
         int pos2 = messages.IndexOf(":", pos1 + 3);
         if (pos2 > -1)
         {
            messages = messages.Remove(0, pos2 + 1).Trim();
         }
         else
         {
            messages = messages.Remove(0, pos1 + 4).Trim();
         }
      }

      // Löschen des StackTrace (der mit "\n   " eingeleitet wird)
      Match match = Regex.Match(messages, "\n   ");
      if (match.Success)
      {
         messages = messages.Remove(match.Index,
            messages.Length - match.Index).Trim();
```

Listing 62: Methode zur Ermittlung aller Nachrichten einer Exception

```
      }
   }
   else
   {
      // Die Nachricht der aktuellen Exception ermitteln
      messages = ex.Message;

      // Die Nachricht(en) der inneren Exception ermitteln
      if (ex.InnerException != null)
      {
         messages += Environment.NewLine +
            GetExceptionMessages(ex.InnerException);
      }

      // Überprüfen, ob Nachrichten eventuell mehrfach vorkommen
      string[] messageList = Regex.Split(messages, Environment.NewLine);
      for (int i = 0; i < messageList.Length; i++)
      {
         string message = messageList[i];
         for (int j = i + 1; j < messageList.Length; j++)
         {
            if (messageList[j] == messageList[i])
            {
               messageList[j] = null;
            }
         }
      }

      // Die übrig gebliebenen Nachrichten wieder zu einem String
      // zusammensetzen
      messages = null;
      for (int i = 0; i < messageList.Length; i++)
      {
         if (messageList[i] != null)
         {
            if (messages != null)
            {
               messages += Environment.NewLine;
            }
            messages += messageList[i];
         }
      }
   }
```

Listing 62: Methode zur Ermittlung aller Nachrichten einer Exception (Forts.)

```
   // Ergebnis zurückgeben
   return messages;
}
```

Listing 62: Methode zur Ermittlung aller Nachrichten einer Exception (Forts.)

46 Exceptions in Anwendungen korrekt auswerten

In einer Anwendung sollten Exceptions idealerweise immer abgefangen werden. In einer Windowsanwendung betrifft das zumindest alle Ereignismethoden, die demnach grundsätzlich mit einem `try-catch`-Block ausgerüstet werden sollten. Da Exceptions in untergeordnet aufgerufenen Methoden nach oben gereicht werden, werden diese spätestens in der Ereignismethode abgefangen und angezeigt.

Wenn Sie Threading einsetzen, sollten Sie alle Thread-Methoden ebenfalls mit einer Fehlerbehandlung ausstatten. Der Grund dafür ist, dass Exceptions, die in einem Arbeits-Thread auftreten, (natürlich) nicht an den UI-Thread weitergereicht werden und – wie Sie gleich noch sehen werden – zu einer unschönen Fehlermeldung mit nachfolgendem Absturz der Anwendung führen.

In meinen Anwendungen zeige ich für unerwartete Fehler meistens die Nachrichten der Exception und ihrer inneren Exceptions an (siehe Rezept 45).

Zusätzlich sollten Sie alle unerwarteten Exceptions noch protokollieren, wozu Sie idealerweise *log4net* verwenden. An Hand der Protokolleinträge können Sie Fehler, die lediglich beim Anwender auftreten und auf Ihrem System nicht nachvollziehbar sind, einfacher lokalisieren. Geben Sie auf jeden Fall den Stack-Trace im Protokoll aus (was log4net mit der Defaultkonfiguration automatisch erledigt). Wenn Sie dem Kunden eine Debug-Version Ihrer Anwendung zusammen mit allen *.pdb*-Dateien (auch denen der verwendeten Klassenbibliotheken) zur Verfügung stellen, erhalten Sie im Stack-Trace sogar die Nummer der Zeile, in der der Fehler aufgetreten ist.

Eine typische Fehlerbehandlung sieht dann etwa so aus:

```
private void btnCatchErrorInEventMethod_Click(object sender, EventArgs e)
{
   StreamReader sr = null;
   string fileName = "Z:\\NonExistingFile.txt";
   try
   {
      // Eine Datei öffnen, die wahrscheinlich nicht existiert
      sr = new StreamReader(fileName);
      string fileContent = sr.ReadToEnd();
   }
   catch (Exception ex)
   {
```

Listing 63: Abfangen von Exceptions in einer Ereignismethode

```
      // Protokollierung des Fehlers
      logger.Error("Fehler beim Öffnen der Datei '" + fileName + "'", ex);

      // Ausgabe der Fehlermeldungen
      MessageBox.Show("Beim Öffnen der Datei '" + fileName +
         "' ist ein Fehler aufgetreten: " +
         ExceptionUtils.GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   finally
   {
      try
      {
         if (sr != null)
         {
            sr.Close();
         }
      }
      catch
      {
         // Fehler beim Schließen des StreamReaders
         // werden ignoriert
      }
   }
}
```

Listing 63: Abfangen von Exceptions in einer Ereignismethode (Forts.)

Dieses Beispiel muss in einem Formular ausgeführt werden, das als privates Feld einen log4net-Logger besitzt, der korrekt konfiguriert ist Die log4net-Assembly muss im Projekt referenziert werden, außerdem sind einige usings notwendig. Schauen Sie sich idealerweise das Beispiel zu diesem Rezept an ☺.

Das Abfangen von Exceptions in jeder Ereignismethode wäre der Idealfall, der in der Praxis leider häufig nicht eintritt. In vielen Fällen vergessen Programmierer das Abfangen oder lassen es weg, weil sie denken, dass in der betreffenden Methode eigentlich ja keine Exceptions auftreten können. Treten dann aber beim Kunden Ausnahmen auf, kann die Fehlersuche sehr aufwändig werden. Unbehandelte Exceptions, die im UI-Thread auftreten, werden noch relativ übersichtlich angezeigt, abgesehen davon, dass der Benutzer entscheiden muss, ob er die Anwendung weiter ausführt oder beendet (was für die meisten Benutzer zu kompliziert ist).



Abbildung 35: Anzeige einer unbehandelten Exception, die im UI-Thread aufgetreten ist

Exceptions, die in einem (Arbeits-)Thread auftreten, der vom UI-Thread aus gestartet wurde, führen aber dummerweise nicht zu einer Anzeige wie bei UI-Thread-Exceptions, sondern (je nach Konfiguration des Windows-Systems) zu einer Windows-Problemmeldung (siehe Abbildung 36). Abgesehen davon, dass Microsoft nicht viel davon hat, wenn der Anwender die Fehlerinformationen dorthin sendet, sind diese Informationen für den Entwickler wenig informativ. Sie enthalten z. B. keine Informationen über die Exception (und damit auch keinen Stack-Trace).



Abbildung 36: Anzeige einer unbehandelten Exception, die in einem Arbeits-Thread aufgetreten ist

Sie sollten diese unbehandelten Exceptions behandeln, damit Sie Fehler zum einen in einer einheitlichen und anwenderfreundlichen Weise anzeigen und zum anderen alle Fehler für die spätere Fehlersuche protokollieren können.

Unbehandelte Ausnahmen, die im UI-Thread auftreten, werden an das `ThreadException`-Ereignis der `Application`-Klasse weitergegeben. Wenn Sie dieses Ereignis behandeln, können Sie alle unbehandelten UI-Thread-Exceptions bearbeiten. Die Default-.NET- Fehlermeldung tritt dann nicht mehr auf.

Unbehandelte Ausnahmen, die in einem Arbeits-Thread auftreten, werden allerdings über das `UnhandledException`-Ereignis der aktuellen Anwendungsdomäne (`AppDomain.CurrentDomain`) ge-

meldet. Sie können die aufgetretene Exception hier aus dem übergebenen Ereignisargument-Objekt auslesen, die Eigenschaft `ExceptionObject` referenziert jedoch nicht eine `Exception`, sondern lediglich ein `Object`. Ein anderes, von mir noch nicht größtes Problem ist, dass die Anwendung auch dann, wenn Sie dieses Ereignis behandeln, abstürzt und der Fehler in der Windows-Problemmeldung gemeldet wird.

Auf jeden Fall haben Sie so aber die Möglichkeit, unbehandelte Exceptions abzufangen, zu protokollieren und dem Benutzer zu melden.

Die Ereignisse weisen Sie idealerweise in der `Main`-Methode Ihrer Anwendung zu:

```
// Das ThreadException-Ereignis der Application-Klasse zuweisen
// um unbehandelte UI-Thread-Fehler abzufangen
Application.ThreadException +=
   new System.Threading.ThreadExceptionEventHandler(
   Program.Application_ThreadException);

// Das UnhandledException-Ereignis der aktuellen Anwendungsdomäne zuweisen
// um unbehandelte Arbeits-Thread-Fehler abzufangen
AppDomain.CurrentDomain.UnhandledException +=
   new UnhandledExceptionEventHandler(
   Program.CurrentDomain_UnhandledException);
```

Listing 64: Zuweisen der Ereignisse zum Abfangen unbehandelter Exceptions

In den Ereignismethoden können Sie die unbehandelten Exceptions nun protokollieren und dem Benutzer melden:

```
/* Fängt unbehandelte Exceptions ab, die im nUI-Thread auftreten */
private static void Application_ThreadException(object sender,
   System.Threading.ThreadExceptionEventArgs e)
{
   // Fehler protokollieren
   logger.Fatal("Unbehandelter UI-Thread-Fehler", e.Exception);

   // Fehler melden
   MessageBox.Show("Unbehandelter Fehler: " +
      ExceptionUtils.GetExceptionMessages(e.Exception),
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
}

/* Fängt unbehandelte Exceptions ab, die in einem Arbeit-Thread auftreten */
private static void CurrentDomain_UnhandledException(object sender,
UnhandledExceptionEventArgs e)
{
```

Listing 65: Ereignismethoden zum Abfangen unbehandelter Exceptions

```
   // Exception auslesen
   Exception ex = e.ExceptionObject as Exception;

   if (ex != null)
   {
      // Fehler protokollieren
      logger.Fatal("Unbehandelter Arbeits-Thread-Fehler", ex);

      // Fehler melden
      MessageBox.Show("Unbehandelter Fehler: " +
         ExceptionUtils.GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   else
   {
      // Fehler protokollieren
      logger.Fatal("Unbehandelter Arbeits-Thread-Fehler vom Typ '" +
      e.ExceptionObject.GetType().Name + "'");

      // Fehler melden
      MessageBox.Show("Unbehandelter Fehler  vom Typ '" +
         e.ExceptionObject.GetType().Name + "'",
         Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 65: Ereignismethoden zum Abfangen unbehandelter Exceptions (Forts.)

Datum und Zeit

47 Aktuelles Datum und aktuelle Uhrzeit ermitteln

Das aktuelle Systemdatum (inklusive Uhrzeit) erhalten Sie über die `Now`-Eigenschaft der `DateTime`-Struktur (aus dem Namensraum `System`):

```
// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;
string currentDate = now.ToShortDateString();
string currentTime = now.ToShortTimeString();

// Datum und Zeit ausgeben
Console.WriteLine("Datum: {0}", currentDate);
Console.WriteLine("Zeit: {0}", currentTime);
```

Listing 66: Ermitteln des aktuellen Systemdatums inklusive der Zeit



Abbildung 37: Ermitteln des aktuellen Systemdatums und der aktuellen Systemzeit

48 Datumswerte formatieren

Datumswerte können Sie über verschiedene Methoden eines `DateTime`-Objekts im Format der aktuell eingestellten Kultur formatieren.

- `ToString` gibt das Datum in einer kurzen Datumsform mit einer kurzen Zeitangabe zurück,
- `ToShortDateString` gibt den Datumsanteil in einer kurzen Form zurück (z. B. "19.05.2003"),
- `ToLongDateString` gibt den Datumsanteil in einer langen Form zurück (z. B. "Montag, 19 Mai 2003"),
- `ToShortTimeString` gibt den Zeitanteil in einer kurzen Form zurück (z. B. "16:14"),
- `ToLongTimeString` gibt den Zeitanteil in einer langen Form zurück (z. B. "16:14:49").

Die Muster, die zur Formatierung verwendet werden, werden in Eigenschaften der `DateTimeFormat`-Eigenschaft des `CultureInfo`-Objekts verwaltet, das dem aktuellen Thread zugeordnet ist (siehe Rezept 49). Diese Eigenschaften heißen `FullDateTimePattern`, `ShortDatePattern`, `LongDatePattern`, `ShortTimePattern` und `LongTimePattern`.

Neben den `DateTime`-Methoden können Sie auch die `Format`-Methode der `String`-Klasse verwenden, der Sie für ein Datum (u. a.) die folgenden Formatzeichen übergeben können:

- *d*: Formatierung im kurzen Datumsformat,
- *D*: Formatierung im langen Datumsformat,
- *t*: Formatierung im kurzen Zeitformat,
- *T*: Formatierung im langen Zeitformat,
- *f*: Formatierung im vollständigen Datumsformat mit kurzer Zeitangabe,
- *F*: Formatierung im vollständigen Datumsformat mit langer Zeitangabe,
- *d*, *dd*, *ddd*, *dddd*: definiert den Tag des Monats als Zahl ohne bzw. mit führender 0 oder in abgekürzter (Mo, Di, Mi) oder voll ausgeschriebener Schreibweise (Montag, Dienstag etc.),
- *M*, *MM*, *MMM*, *MMMM*: definiert den Monat als Zahl ohne bzw. mit führender 0 oder in abgekürzter (Jan, Feb, Mrz) oder voll ausgeschriebener Schreibweise (Januar, Februar etc.),
- *yy*, *yyyy*: bestimmt, dass das Jahr zwei- bzw. vierstellig ausgegeben wird,
- *H*, *HH*: bestimmt, dass die Stunde im 24-Stunden-Format ohne oder mit führender 0 ausgegeben wird. Beachten Sie, dass ein kleingeschriebenes *h* die Stunde im 12-Stunden-Format formatiert,
- *m*, *mm*: definiert, dass die Minute ohne oder mit führender 0 ausgegeben wird,
- *s*, *ss*: bestimmt, dass die Sekunde ohne oder mit führender 0 ausgegeben wird.

Listing 67 zeigt einige typische Datumsformatierungen. Das Programm erfordert den Import des Namensraums `System`.

```
DateTime now = DateTime.Now;

Console.WriteLine("ToString:\r\n{0}\r\n", now.ToString());
Console.WriteLine("ToShortDateString:\r\n{0}\r\n",
   now.ToShortDateString());
Console.WriteLine("ToLongDateString:\r\n{0}\r\n",
   now.ToLongDateString());
Console.WriteLine("ToShortTimeString:\r\n{0}\r\n",
   now.ToShortTimeString());
Console.WriteLine("ToLongTimeString:\r\n{0}\r\n",
   now.ToLongTimeString());
Console.WriteLine(String.Format("dd.MM.yy:\r\n{0:dd.MM.yy}\r\n", now));
Console.WriteLine(String.Format("dddd, dd. MMMM yy:\r\n" +
   "{0:dddd, dd. MMMM yyyy}\r\n", now));
Console.WriteLine(String.Format("ddd, dd. MMM yy:\r\n" +
   "{0:ddd, dd. MMM yyyy}\r\n", now));
Console.WriteLine(String.Format("HH:mm:ss:\r\n" +
   "{0:HH:mm:ss}", now));
```

Listing 67: Einige typische Datumsformatierungen

49 Datumswerte in verschiedenen Länderformaten ausgeben

Wenn Sie die `To`-Methoden eines `DateTime`-Objekts (`ToString`, `ToShortDateString` etc.) aufrufen, werden Datumswerte immer entsprechend der aktuellen Kultur formatiert. Leider besitzen diese Methoden kein Argument, an dem ein `CultureInfo`-Objekt für eine spezielle Kultur übergeben werden kann. Wollen Sie diese Methoden verwenden um ein Datum in einem spezifischen Länderformat auszugeben, müssen Sie die aktuelle Kultur umstellen.

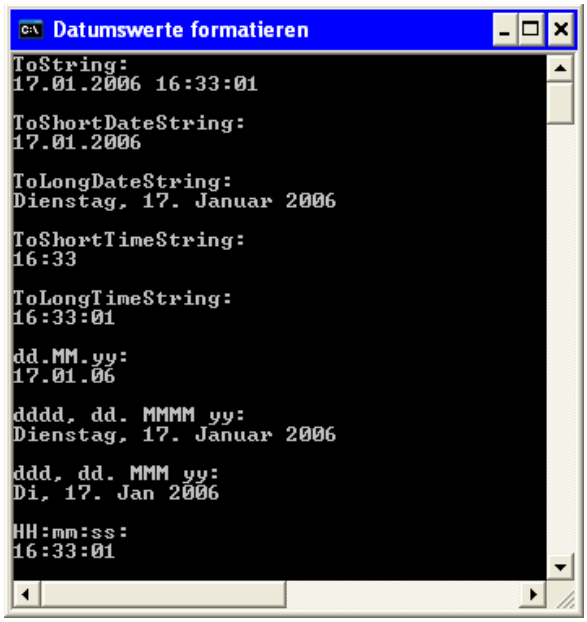


Abbildung 38: Typische Datumsformatierungen auf einem deutschen Rechner

Einfacher ist es jedoch, die `Format`-Methode der `String`-Klasse zu verwenden. Dieser Methode können Sie am ersten Argument ein Objekt übergeben, das die `IFormatProvider`-Schnittstelle implementiert. Wenn Sie kein eigenes Formatier-Objekt implementieren wollen (was wohl nur selten Sinn macht), können Sie dazu ein `CultureInfo`-Objekt verwenden, das die Formatierung dann entsprechend seiner Kultur bestimmt. Da die `Format`-Methode auch Formatierzeichen für festgelegte Formate besitzt (z. B. *d* für ein kurzes Datumsmuster), können Sie sehr flexibel formatieren.

Die aktuelle Kultur ist definiert über ein `CultureInfo`-Objekt, das der Eigenschaft `CurrentCulture` des aktuellen Thread zugewiesen ist. Das `Thread`-Objekt des aktuellen Thread können Sie über `System.Threading.Thread.CurrentThread` auslesen. Um ein Datum nun über die `To`-Methoden einer `DateTime`-Instanz in einem bestimmten Länderformat auszugeben, müssen Sie dem aktuellen Thread ein spezielles `CultureInfo`-Objekt zuweisen.

Ein `CultureInfo`-Objekt für eine spezielle Kultur können Sie über die statische `CreateSpecificCulture`-Methode der `CultureInfo`-Klasse (aus dem Namensraum `System.Globalization`) erzeugen. Dieser Methode übergeben Sie eine der international genormten Kultur-Kurzbezeichnungen.

Tabelle 8 stellt einige wichtige dieser Kurzbezeichnungen dar. Die gesamte Liste finden Sie in der Dokumentation der `CultureInfo`-Klasse.

Kurzbezeichnung	Kultur
de	Deutsch allgemein
de-DE	Deutsch in Deutschland
de-AT	Österreichisches Deutsch
de-CH	Schweizer Deutsch
fr	Französisch allgemein
it	Italienisch allgemein
es	Spanisch allgemein
en	Englisch allgemein
en-US	US-amerikanisches Englisch
en-GB	Englisch in Großbritannien
en-AU	Englisch in Australien

Tabelle 8: Einige wichtige Kultur-Kurzbezeichnungen

Das folgende Beispiel zeigt eine Formatierung im vollen Datumsformat mit langer Zeit für die australische Kultur:

```
string culture = "en-AU";
string format = "G";
string formattedDate = String.Format(
   CultureInfo.CreateSpecificCulture(culture),
   "{0:" + format + "}" , DateTime.Now);
```

Listing 68: Ausgabe des aktuellen Datums im australischen Format

Zur Kompilierung dieses Beispiels müssen Sie den Namensraum `System.Globalization` importieren.

Tabelle 9 zeigt die für Datumswerte möglichen Formatierzeichen.

Zeichen	Bedeutung
d	Datum im kurzen Format
D	Datum im langen Format
t	Zeit im kurzen Format (ohne Sekunden)
T	Zeit im langen Format (inkl. Sekunden)
f	Komplettes Datum (langes Datum, kurze Zeit)
F	Komplettes Datum (langes Datum, lange Zeit)
g	Komplettes Datum (kurzes Datum, kurze Zeit)

Tabelle 9: Die Formatier-Zeichen für Datumswerte

Zeichen	Bedeutung
G	Komplettes Datum (kurzes Datum, lange Zeit)
m oder M	Nur Monat und Tag
r oder R	RFC1123-Format
s	ISO-8601-Format
u	UT-Datum (Universal Time) im UT-Format
U	Datum im langen Datumsformat, allerdings wird das universelle Datum (UT) ausgegeben

Tabelle 9: Die Formatier-Zeichen für Datumswerte (Forts.)

Alternativ können Sie, wie bereits gesagt, die `To`-Methoden der `DateTime`-Klasse verwenden. Listing 69 zeigt, wie Sie auf diese Weise ein Datum z. B. im englischen Standardformat ausgeben. Das Programm verwendet die `ToLongDateString`-Methode eines `DateTime`-Objekts zur Formatierung. Es referenziert zunächst die aktuell eingestellte Kultur in der Variablen `currentCulture`, damit diese später wieder zurückgesetzt werden kann. Dann erzeugt das Programm eine `CultureInfo`-Instanz für die allgemeine englische Kultur und weist diese dem aktuellen Thread zu. Nach der Formatierung des Datumswerts wird die zuvor eingestellte Kultur dann wieder zurückgeschrieben.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Globalization` und `System.Threading` importieren.

```
try
{
   // Aktuelle Kultur auf die englische umschalten
   string culture = "en";
   Thread.CurrentThread.CurrentCulture =
      CultureInfo.CreateSpecificCulture(culture);

   // Datumswert formatieren und ausgeben
   formattedDate = DateTime.Now.ToLongDateString();
   Console.WriteLine(formattedDate);
}
finally
{
   // Aktuelle Kultur wieder zurücksetzen
   Thread.CurrentThread.CurrentCulture = currentCulture;
}
```

Listing 69: Ausgabe des aktuellen Datums im US-englischen Format

Abbildung 39 zeigt die Ausgabe beider Beispiele an der Konsole.



Abbildung 39: Das aktuelle Datum im US-englischen Format

50 Datum in das ISO-Format konvertieren

Das Schweizer ISO-Institut (International Organization for Standardization), das ähnlich dem deutschen DIN-Institut (internationale) Standards entwickelt und veröffentlicht, hat in der ISO-Norm 8601 *(www.iso.ch/iso/en/prods-services/popstds/datesandtime.html)* einen Standard für die Übermittlung von Datumswerten geschaffen. Dieser Standard ist sehr interessant für die Angabe von Datumswerten in SQL-Anweisungen, da viele Datenbanksysteme (z. B. MySQL und der Microsoft SQL Server) neben einem proprietären Format auch das ISO-Format unterstützen. Wenn Sie dieses Format verwenden, müssen Sie sich keine Gedanken um das spezielle Format des Datenbanksystems machen. Aber auch bei der Übertragung von Daten in Form von XML- oder Textdateien kann das ISO-Datum dabei helfen, dass unterschiedliche Systeme keine Probleme mit Datumswerten haben.

ISO 8601 sieht vor, dass ein Datum im folgenden Format angegeben wird:

```
YYYY-MM-DD
```

Alternativ können Sie das folgende Format verwenden, bei dem die Kalenderwoche an Stelle des Monats und der Wochentag an Stelle des Tags steht:

```
YYYY-Www-D
```

Eine Zeitangabe ist folgendermaßen definiert:

```
hh:mm:ss
```

Sie können beide Angaben kombinieren, wobei nach der Norm ein *T* zwischen Datum und Zeit stehen muss (viele Systeme nehmen an Stelle des *T* aber auch ein Leerzeichen an). Das Datum 1.6.2005 10:50:00 wird z. B. so angegeben:

```
2005-06-01T10:50:00
```

Die Datumsangabe wird häufig mit einem großen Z abgeschlossen, das definiert, dass es sich um ein universelles Datum handelt, das auch als *Zulu Time* bezeichnet wird:

```
2005-06-01T10:50:00Z
```

Das Z ist allerdings kein Bestandteil der ISO-Norm.

Prinzipiell könnten Sie die `Format`-Methode der `String`-Klasse verwenden, um ein Datum im ISO-Format auszugeben:

```
string formattedDate = String.Format("{0:u}", DateTime.Now)
```

Der zurückgegebene String (Beispiel: »2005-10-10 15:49:51Z«) entspricht jedoch leider nicht dem vom ISO-Institut vorgegebenen Format, da das T zwischen dem Datum und der Zeit fehlt

und der String mit einem Z abgeschlossen ist. Dies könnte unter Umständen in einigen Fällen zu Problemen führen.

Deshalb habe ich eine eigene Methode `ToIsoDateTime` entwickelt, die die `Format`-Methode der `String`-Klasse mit eigenen Formatier-Angaben einsetzt. Das Argument `includeTime` bestimmt dabei, ob die Zeit mit eingeschlossen werden soll.

```
public static string ToIsoDateTime(this DateTime date,
   bool includeTime)
{
   if (includeTime)
   {
      return String.Format("{0:yyyy-MM-ddTHH:mm:ss}", date);
   }
   else
   {
      return String.Format("{0:yyyy-MM-dd}", date);
   }
}
```

Listing 70: Methode zur Ermittlung eines ISO-Datums

`ToIsoDateTime` erfordert den Import des Namensraums `System`.

51 Datumswerte vergleichen

`DateTime`-Instanzen können Sie direkt miteinander vergleichen, da die `DateTime`-Struktur die Vergleichsoperatoren überlädt. Diese Operatoren berücksichtigen neben dem Datumsteil natürlich auch den Zeitanteil. Das Datum 01.01.2004 10:00:00 ist demnach größer als das Datum 01.01.2004 09:59:59. Listing 71 zeigt einige typische Vergleiche von Datumswerten. Das Programm erfordert den Import des Namensraums `System`.

```
// DateTime-Instanzen erzeugen
DateTime d1 = new DateTime(2003, 12, 31);
DateTime d2 = new DateTime(2004, 1, 1);
DateTime d3 = new DateTime(2004, 1, 1);
DateTime d4 = new DateTime(2004, 1, 1, 12, 0, 0);

if (d1 < d2)
{
   Console.WriteLine("{0} ist kleiner als {1}", d1, d2);
}
if (d2 == d3)
{
   Console.WriteLine("{0} ist gleich {1}", d2, d3);
```

Listing 71: Vergleichen von Datumswerten

```
}
if (d3 != d4)
{
   Console.WriteLine("{0} ist ungleich {1}", d3, d4);
}
if (d4 > d3)
{
   Console.WriteLine("{0} ist größer als {1}", d4, d3);
}
```

Listing 71: Vergleichen von Datumswerten (Forts.)



Abbildung 40: Das Beispielprogramm zum Vergleichen von Datumswerten

52 Die Differenz zwischen zwei Datumswerten berechnen

Die Differenz zwischen zwei Datumswerten können Sie einfach über eine Subtraktion des kleineren vom größeren Datumswert ermitteln. Eine Subtraktion zweier `DateTime`-Werte (aus dem Namensraum `System`) ergibt ein `TimeSpan`-Objekt. Über die Eigenschaften `Ticks` (Anzahl der 100-Nanosekunden-Intervalle zwischen den Datumswerten), `TotalDays`, `TotalHours`, `TotalMinutes`, `TotalSeconds` und `TotalMilliseconds` können Sie die Differenz zwischen den beiden Datumswerten auslesen:

```
// Zwei DateTime-Objekte erzeugen
DateTime date1 = DateTime.Now;
DateTime date2 = new DateTime(2006, 1, 1);

Console.WriteLine("Datum 1: {0}", date1);
Console.WriteLine("Datum 2: {0}", date2);
Console.WriteLine();

// Berechnen der Differenz
TimeSpan dateDiff = date2 - date1;

// Ausgabe der Differenz mit dem TimeSpan-Objekt
Console.WriteLine("Differenz:");
Console.WriteLine("{0} Ticks", dateDiff.Ticks);
```

Listing 72: Berechnen der Differenz zwischen zwei Datumswerten über ein TimeSpan-Objekt

```
Console.WriteLine("{0} Tage", dateDiff.TotalDays);
Console.WriteLine("{0} Stunden", dateDiff.TotalHours);
Console.WriteLine("{0} Minuten", dateDiff.TotalMinutes);
Console.WriteLine("{0} Sekunden", dateDiff.TotalSeconds);
Console.WriteLine("{0} Millisekunden", dateDiff.TotalMilliseconds);
```

Listing 72: Berechnen der Differenz zwischen zwei Datumswerten über ein TimeSpan-Objekt (Forts.)

Wenn Sie die Differenz zwischen zwei Datumswerten in Einheiten berechnen wollen, die größer sind als Tage (also in Wochen, Monaten, Quartalen oder Jahren), erhalten Sie das Ergebnis nicht direkt über ein `TimeSpan`-Objekt. Der Grund dafür ist, dass solche Berechnungen davon abhängen, an welchem Datum sie beginnen und enden und wie lang das jeweilige Intervall ist. Bei der Berechnung der Anzahl der Monate zwischen zwei Datumswerten muss z. B. die unterschiedliche Länge der einzelnen Monate berücksichtigt werden.

Für eine solche Berechnung können Sie eine kleine Anleihe bei Visual Basic machen: Im Modul `DateAndTime` finden Sie die Methode `DateDiff`, die die Differenz zwischen zwei Datumswerten in Einheiten eines gegebenen Intervalls berechnet. Diese Funktionalität steht in den »normalen« .NET-Klassen leider nicht zur Verfügung, was Sie auch daran erkennen können, dass (bewiesen über den .NET-Reflector von Lutz Roeder) `DateDiff` eine in `DateAndTime` hartcodierte Methode ist.

Referenzieren Sie die Assembly *Microsoft.VisualBasic.dll*, wenn Sie `DateDiff` ausprobieren wollen. Dabei sollten Sie aber beachten, dass diese Methode bei der Berechnung von Jahren, Quartalen und Monaten immer ermittelt, wie viele Intervalle zwischen zwei Datumswerten liegen, wenn nur die jeweiligen Intervall-Teile des Datums berücksichtigt werden. Wenn Sie z. B. die Jahres-Differenz zwischen dem 31.12.2005 und dem 1.1.2006 berechnen, erhalten Sie als Ergebnis 1, weil zwischen 2005 und 2006 ein Jahr liegt. Berechnen Sie die Anzahl Quartale, erhalten Sie ebenfalls 1, weil zwischen dem Quartal 4/2005 und dem Quartal 1/2006 ein Quartal liegt. Ähnliches gilt für die Berechnung der Monats-Differenz, da zwischen dem Monat 12/2005 und dem Monat 1/2006 ein Monat liegt.

In meinen Augen ist eine solche Berechnung kaum zu gebrauchen, weswegen ich diese hier auch nicht weiter behandle. Was ich in meiner Praxis häufiger benötige, ist eine Berechnung der tatsächlichen Differenz in einem gegebenen Intervall, inklusive der Nachkommastellen. Ein Beispiel dafür ist die Ermittlung des Alters einer Person in Jahren. Bei dieser Berechnung liegen zwischen dem 31.12.2005 und dem 1.1.2006 ca. 0,00274 (1 / 365) Jahre. Deshalb habe ich Methoden entwickelt, die die Differenz zwischen zwei Datumswerten in Jahren, Monaten und Wochen korrekt berechnen. Die Methoden gehen prinzipiell so vor, dass sie den Anteil der Intervalle berechnen, die durch das Start- und das Enddatum definiert sind, und die ganzen Intervalle, die dazwischen liegen, aufaddieren. Demnach liegen zwischen dem 01.11.2006 00:00 und dem 15.04.2008 12:00 aufgerundet 0,1671 Jahre (2006) + 1 Jahr (2007) + 0,2883 Jahre (2008). Im Ergebnis sind das dann 1,4554 Jahre.

Die Methode `GetYearDifference`, die die Differenz zwischen zwei Datumswerten in Jahren berechnet, überprüft dazu zunächst, ob beide Datumswerte im selben Jahr liegen. Ist dies der Fall, teilt sie einfach die Anzahl Ticks zwischen den Datumswerten durch die Anzahl der Ticks, die dieses Jahr umspannt.

Liegen beide Datumswerte in unterschiedlichen Jahren, wird zunächst überprüft, ob das erste Datum größer ist als das zweite, weil in diesem Fall die Datumswerte vertauscht und das Ergebnis negiert werden müssen. Dann berechnet die Methode den Anteil am Jahr vom Startdatum bis zum Ende des Jahres dieses Datums und den entsprechenden Anteil vom Anfang des Jahres des letzten Datums bis zu diesem Datum. Schließlich werden auch die dazwischen liegenden vollen Jahre aufaddiert. Das Ergebnis wird gegebenenfalls negiert zurückgegeben.

```
public static double GetYearDifference(DateTime firstDate,
   DateTime lastDate)
{
   // Differenz berechnen
   TimeSpan timeSpan = lastDate - firstDate;

   if (firstDate.Year == lastDate.Year)
   {
      // Wenn beide Datumswerte im selben Jahr liegen:
      // Die Ticks in diesem Jahr berechnen
      long yearTicks = (
         (new DateTime(firstDate.Year + 1, 1, 1)) -
         (new DateTime(firstDate.Year, 1, 1))).Ticks - 1;

      // Den Anteil an Ticks für das Jahr berechnen
      return timeSpan.Ticks / (double)yearTicks;
   }
   else
   {
      // Wenn beide Datumswerte in unterschiedlichen
      // Jahren liegen:

      // Wenn das erste Datum größer ist als das
      // zweite, werden die Datumswerte vertauscht
      // und das Programm merkt sich, dass das
      // Ergebnis negiert werden muss
      bool negate = false;
      if (firstDate > lastDate)
      {
         DateTime temp = firstDate;
         firstDate = lastDate;
         lastDate = temp;
         negate = true;
      }

      // Den Anteil im ersten Jahr berechnen
      DateTime firstYearStartDate = new DateTime(
         firstDate.Year, 1, 1);
```

Listing 73: Methode zur Berechnung der Jahres-Differenz zwischen zwei Datumswerten

```
      DateTime firstYearEndDate = new DateTime(
         firstDate.Year + 1, 1, 1).AddTicks(-1);
      long yearTicks = (firstYearEndDate -
         firstYearStartDate).Ticks;
      long ticks = (firstYearEndDate - firstDate).Ticks;
      double years = ticks / (double)yearTicks;

      // Den Anteil im letzten Jahr berechnen
      DateTime lastYearStartDate =
         new DateTime(lastDate.Year, 1, 1);
      DateTime lastYearEndDate =
         new DateTime(lastDate.Year + 1, 1, 1).AddTicks(-1);
      yearTicks = (lastYearEndDate - lastYearStartDate).Ticks;
      ticks = (lastDate - lastYearStartDate).Ticks;
      years += ticks / (double)yearTicks;

      // Die Jahre dazwischen addieren und das Ergebnis
      // zurückgeben
      return (years + lastDate.Year - firstDate.Year - 1)
         * (negate ? -1 : 1);
   }
}
```

Listing 73: Methode zur Berechnung der Jahres-Differenz zwischen zwei Datumswerten (Forts.)

`GetYearDifference` erfordert, wie alle weiteren in diesem Rezept beschriebenen Methoden, lediglich das Einbinden des Namensraums `System`.

Die Methode `GetMonthDifference` berechnet die Differenz in Monaten auf eine ähnliche Weise:

```
public static double GetMonthDifference(DateTime firstDate,
   DateTime lastDate)
{
   // Differenz berechnen
   TimeSpan timeSpan = lastDate - firstDate;

   if (firstDate.Year == lastDate.Year &&
      firstDate.Month == lastDate.Month)
   {
      // Wenn beide Datumswerte im selben Monat liegen:
      // Die Ticks in diesem Monat berechnen
      DateTime monthStartDate = new DateTime(
         firstDate.Year, firstDate.Month, 1);
      DateTime monthEndDate = monthStartDate.AddMonths(1).AddTicks(-1);
```

Listing 74: Methode zur Berechnung der Monats-Differenz zwischen zwei Datumswerten

```
      long monthTicks = (monthEndDate - monthStartDate).Ticks;

      // Den Anteil an Ticks für den Monat berechnen
      return timeSpan.Ticks / (double)monthTicks;
   }
   else
   {
      // Wenn beide Datumswerte in unterschiedlichen
      // Monaten liegen:

      // Wenn das erste Datum größer ist als das
      // zweite, werden die Datumswerte vertauscht
      // und das Programm merkt sich, dass das
      // Ergebnis negiert werden muss
      bool negate = false;
      if (firstDate > lastDate)
      {
         DateTime temp = firstDate;
         firstDate = lastDate;
         lastDate = temp;
         negate = true;
      }

      // Den Anteil im ersten Monat berechnen
      DateTime firstMonthStartDate = new DateTime(
         firstDate.Year, firstDate.Month, 1);
      DateTime firstMonthEndDate =
         firstMonthStartDate.AddMonths(1).AddTicks(-1);
      long monthTicks = (firstMonthEndDate - firstMonthStartDate).Ticks;
      long ticks = (firstMonthEndDate - firstDate).Ticks;
      double months = ticks / (double)monthTicks;

      // Den Anteil im letzten Monat berechnen
      DateTime lastMonthStartDate = new DateTime(
         lastDate.Year, lastDate.Month, 1);
      DateTime lastMonthEndDate =
         lastMonthStartDate.AddMonths(1).AddTicks(-1);
      monthTicks = (lastMonthEndDate - lastMonthStartDate).Ticks;
      ticks = (lastDate - lastMonthStartDate).Ticks;
      months += ticks / (double)monthTicks;

      // Die Monate dazwischen addieren
      // und das Ergebnis zurückgeben
      return (months +
         (12 - firstDate.Month) +
```

Listing 74: Methode zur Berechnung der Monats-Differenz zwischen zwei Datumswerten (Forts.)

```
         ((lastDate.Year - firstDate.Year - 1) * 12) +
         (lastDate.Month - 1)) * (negate ? -1 : 1);
   }
}
```

Listing 74: Methode zur Berechnung der Monats-Differenz zwischen zwei Datumswerten (Forts.)

Diese Methode `GetWeekDifference` ist schließlich sehr einfach implementiert, da sie die Differenz in Tagen einfach durch sieben teilen kann.

```
public static double GetWeekDifference(DateTime firstDate,
   DateTime lastDate)
{
   // Differenz in Tagen berechnen und durch
   // 7 geteilt zurückgeben
   TimeSpan timeSpan = lastDate - firstDate;
   return timeSpan.TotalDays / 7;
}
```

Listing 75: Methode zur Berechnung der Wochen-Differenz zwischen zwei Datumswerten

Um die Berechnung von Differenzen in verschiedenen Intervallen auf eine einfache Weise zu ermöglichen, habe ich eine Methode `DateDiff` entwickelt, die die oben beschriebenen Methoden aufruft. Diese Methode arbeitet mit einem Argument vom Typ `DateInterval`, das das Intervall beschreibt, für das das Ergebnis berechnet werden soll.

```
/* Aufzählung der möglichen Datumsintervalle */
public enum DateInterval
{
   Years,
   Months,
   Weeks,
   Days,
   Hours,
   Minutes,
   Seconds,
   Milliseconds
}

/* Berechnet die Differenz zwischen zwei Datumswerten */
public static double DateDiff(DateTime firstDate,
   DateTime lastDate, DateInterval interval)
{
```

Listing 76: Allgemeine Methode zur Berechnung der Differenz zwischen zwei Datumswerten

```
   // Je nach Intervall das Ergebnis berechnen
   switch (interval)
   {
      case DateInterval.Years:
         return GetYearDifference(firstDate, lastDate);

      case DateInterval.Months:
         return GetMonthDifference(firstDate, lastDate);

      case DateInterval.Weeks:
         return GetWeekDifference(firstDate, lastDate);

      case DateInterval.Days:
         return (lastDate - firstDate).TotalDays;

      case DateInterval.Hours:
         return (lastDate - firstDate).TotalHours;

      case DateInterval.Minutes:
         return (lastDate - firstDate).TotalMinutes;

      case DateInterval.Seconds:
         return (lastDate - firstDate).TotalSeconds;

      default:
         return (lastDate - firstDate).TotalMilliseconds;
   }
}
```

Listing 76: Allgemeine Methode zur Berechnung der Differenz zwischen zwei Datumswerten (Forts.)

Abbildung 41 zeigt die Ausgabe eines Programms, das diese Methode anwendet.



Abbildung 41: Datumsdifferenzberechnung

53 Alter in ganzen Jahren berechnen

Mit Rezept 52 können Sie das Alter einer Person oder eines Gegenstands in Jahren berechnen, die dort vorgestellte Methode ist für Fälle, bei denen ein Alter nur in ganzen Jahren berechnet werden soll, jedoch u. U. zu aufwändig.

Deswegen habe ich die Methode `GetAge`, die ich in der ersten Auflage des Codebook entwickelt habe, wieder in das Codebook aufgenommen. `GetAge` berechnet zunächst die Differenz der Jahresangabe in den zwei übergebenen Datumswerten. Danach wird überprüft, ob das größere Datum bezogen auf den Tag und den Monat größer oder gleich dem kleineren Datum ist. Ist dies nicht der Fall, zieht `GetAge` von der Jahresdifferenz ein Jahr ab, da der »Geburtstag« noch nicht erreicht ist.

`GetAge` erfordert den Import des Namensraums `System`.

```
/* Berechnet ein Alter in Jahren */
public static int GetAge(this DateTime lowerDate, DateTime higherDate)
{
   return GetAge(new DateTimeOffset(lowerDate), new DateTimeOffset(higherDate));
}

/* Berechnet ein Alter in Jahren */
public static int GetAge(this DateTimeOffset lowerDate,
   DateTimeOffset higherDate)
{
   // Basis-Alter als Differenz zwischen den Jahren ermitteln
   int age = higherDate.Year - lowerDate.Year;

   // Wenn der Monat des größeren Datums kleiner oder wenn der Monat gleich
   // und der Tag kleiner ist, muss ein Jahr abgezogen werden
   if ((higherDate.Month < lowerDate.Month) ||
      (higherDate.Month == lowerDate.Month && higherDate.Day < lowerDate.Day))
   {
      age--;
   }

   return age;
}
```

Listing 77: Berechnen eines Alters in Jahren

54 Datumswerte addieren und subtrahieren

Häufig ist es notwendig, zu einem gegebenen Datum eine bestimmte Anzahl Datumseinheiten (Minuten, Stunden, Tage, Wochen etc.) zu addieren oder davon abzuziehen. Die `DateTime`-Struktur bietet dazu entsprechende Methoden (Tabelle 10).

Methode	Addiert bzw. subtrahiert …
`DateTime Add(TimeSpan value)`	einen `TimeSpan`-Wert
`DateTime AddDays(double value)`	Tage
`DateTime AddHours(double value)`	Stunden
`DateTime AddMilliseconds(double value)`	Millisekunden
`DateTime AddMinutes(double value)`	Minuten
`DateTime AddMonths(int months)`	Monate
`DateTime AddSeconds(double value)`	Sekunden
`DateTime AddTicks(long value)`	Ticks (100ns-Intervalle)
`DateTime AddYears(int value)`	Jahre

Tabelle 10: Die Methoden der DateTime-Struktur zum Addieren und Subtrahieren von Datumseinheiten

Da Sie den Methoden auch negative Werte übergeben können, können Sie einem Datum auch Einheiten abziehen. Die meisten `Add`-Methoden arbeiten mit `double`-Argumenten, sodass Sie auch Bruchteile der Einheit addieren oder subtrahieren können. Bei Monaten und Jahren (die eine unterschiedliche Anzahl an Tagen besitzen) und bei Ticks ist das prinzipiell nicht möglich, weswegen am Argument dieser Methoden `int`- bzw. `long`-Werte erwartet werden.

Listing 78 zeigt die Anwendung dieser Methoden. Zum Kompilieren des Beispiels müssen Sie den Namensraum `System` importieren.

```
// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;
Console.WriteLine("Jetzt: {0}", now.ToString());

// 12,5 Stunden abziehen
DateTime date = now.AddHours(-12.5);
Console.WriteLine("Jetzt - 12,5 Stunden: {0}", date.ToString());

// 10,5 Tage hinzuaddieren
date = now.AddDays(10.5);
Console.WriteLine("Jetzt + 10,5 Tage: {0}", date.ToString());

// Sechs Monate hinzuaddieren
date = now.AddMonths(6);
Console.WriteLine("Jetzt + 6 Monate: {0}", date.ToString());

// Zwei Jahre abziehen
date = now.AddYears(-2);
Console.WriteLine("Jetzt - 2 Jahre: {0}", date.ToString());
```

Listing 78: Addieren und Subtrahieren von Datumseinheiten

```
// Mein Alter als TimeSpan-Wert berechnen (natürlich mit
// gelogenem Geburtsdatum ;-))
DateTime birthDate = new DateTime(1972, 9, 27, 2, 30, 0);
Console.WriteLine("Mein (gelogenes) Geburtsdatum: {0}",
   birthDate.ToString());
TimeSpan age = now - birthDate;

// Damit errechnen, welches Datum meinem doppelten Alter entspricht
date = now.Add(age);
Console.WriteLine("Dann bin ich doppelt so alt wie jetzt: {0}",
   date.ToString());
```

Listing 78: Addieren und Subtrahieren von Datumseinheiten (Forts.)

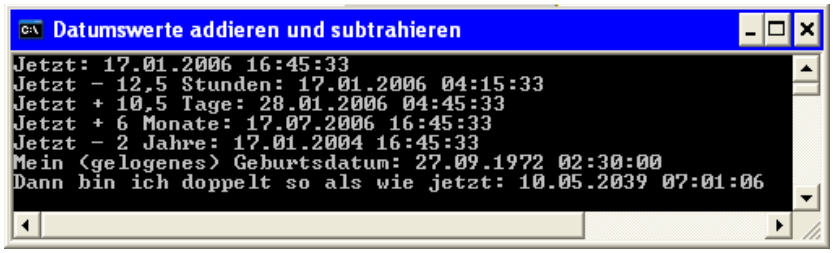


Abbildung 42: Die Datumsberechnung

55 Eine Zeitangabe in einem String in ein Datum mit der Basis DateTime.Min umwandeln

Wenn Sie eine Zeitangabe, die in einem String gespeichert ist, über `Convert.ToDateTime` in ein Datum umwandeln, erhalten Sie ein Datum, das das aktuelle Datum im Datumsteil speichert. In manchen Fällen ist dies aber hinderlich, z.B. wenn eigentlich nur Zeitangaben gespeichert, diese gegebenenfalls an unterschiedlichen Tagen eingegeben und irgendwann verglichen werden sollen. In diesem Fall wäre es besser, den String so umzuwandeln, dass das kleinste Datum als Datumsbasis verwendet wird. Dies erreichen Sie, indem Sie die `DateTime.Parse`-Methode verwenden. Am ersten Argument übergeben Sie den zu konvertierenden String. Am dritten Argument können Sie Werte der `System.Globalization.DateTimeStyles`-Aufzählung übergeben, die die Art der Formatierung bestimmen. Der Wert `NoCurrentDateDefault` steht dafür, dass für Datumsstrings ohne Datumsangabe nicht das aktuelle Datum, sondern das Basisdatum (1.1.0001) verwendet wird. Da keine Variante der `Parse`-Methode existiert, die neben dem zu konvertierenden String nur dieses Argument erwartet, müssen Sie die Variante verwenden, der am zweiten Argument ein `IFormatProvider` übergeben wird, der für die Formatierung des String zuständig ist. Hier können Sie mit `System.Globalization.DateTimeFormatInfo.CurrentInfo` den aktuellen Format-Provider angeben.

Zum Kompilieren des folgenden Beispiels müssen Sie die Namensräume `System` und `System.Globalization` einbinden.

```
string timeString = "10:00";

DateTime date = DateTime.Parse(timeString, DateTimeFormatInfo.CurrentInfo,
   DateTimeStyles.NoCurrentDateDefault);
```

Listing 79: Konvertieren eines Strings mit einer Zeitangabe in ein Datum mit dem kleinsten Datum als Basis

56 Die Anzahl der Tage eines Monats berechnen

Die Anzahl der Tage eines Monats können Sie über die Methode `DaysInMonth` der `DateTime`-Struktur berechnen. Dieser Methode übergeben Sie am ersten Argument das Jahr und am zweiten den Monat. Das Beispiel in Listing 80, das den Import des `System`-Namensraums erfordert, berechnet die Anzahl der Tage des aktuellen Monats.

```
DateTime now = DateTime.Now;
int daysInMonth = DateTime.DaysInMonth(now.Year, now.Month);

Console.WriteLine("Tage des Monats {0}/{1}: {2}",
   now.Month, now.Year, daysInMonth);
```

Listing 80: Berechnen der Anzahl der Tage des aktuellen Monats



Abbildung 43: Das Beispielprogramm in Aktion

57 Die Anzahl der Tage eines Jahres berechnen

Die Anzahl der Tage eines Jahres können Sie nicht wie bei der Anzahl der Tage eines Monats über eine Methode der `DateTime`-Struktur berechnen. Dazu müssen Sie ein Objekt einer von `Calendar` abgeleiteten Klasse verwenden, das den Kalender des Systems repräsentiert (bei uns ist das der gregorianische Kalender). Den aktuellen Kalender können Sie über die `Calendar`-Eigenschaft des `CultureInfo`-Objekts, das die aktuelle Kultur darstellt (`System.Globalization.CultureInfo.CurrentCulture.Calendar`), ermitteln. Die Anzahl der Tage in einem Jahr erhalten Sie dann über die Methode `GetDaysInYear`, der Sie das Jahr übergeben, dessen Tage Sie ermitteln wollen.

Listing 81 zeigt dies für das aktuelle Jahr. Das Beispiel erfordert den Import der Namensräume `System` und `System.Globalization`.

```
// GregorianCalendar-Objekt für den aktuellen Kalender erzeugen
GregorianCalendar calendar = new GregorianCalendar();

// Anzahl der Tage im aktuellen Jahr berechnen
DateTime now = DateTime.Now;
int daysInYear = calendar.GetDaysInYear(now.Year);

Console.WriteLine("Tage des Jahres {0}: {1}",
   now.Year, daysInYear);
```

Listing 81: Berechnen der Anzahl der Tage eines Jahres

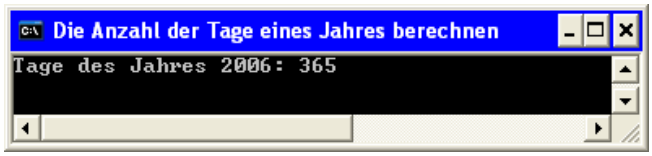


Abbildung 44: Das Beispielprogramm in Aktion

58 Den Wochentag eines Datums ermitteln

Den Wochentag eines Datums erhalten Sie über die `DayOfWeek`-Eigenschaft der `DateTime`-Instanz. Diese Eigenschaft verwaltet einen Wert der `DayOfWeek`-Aufzählung mit den Konstanten `Sunday`, `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday` und `Saturday`. Die Konstanten beginnen übrigens bei `Sunday`, da in den USA die Woche mit dem Sonntag beginnt.

Listing 82 zeigt, wie Sie den Wochentag eines gegebenen Datums ermitteln. Zum Kompilieren dieses Beispiels müssen Sie den Namensraum `System` einbinden.

```
// Datum festlegen
DateTime date = DateTime.Now;

// Wochentag ermitteln
switch (date.DayOfWeek)
{
   case DayOfWeek.Monday:
      Console.WriteLine("Heute is Montag");
      break;

   case DayOfWeek.Tuesday:
      Console.WriteLine("Heute is Dienstag");
      break;

   case DayOfWeek.Wednesday:
      Console.WriteLine("Heute is Mittwoch");
```

Listing 82: Ermitteln des Wochentags eines gegebenen Datums

```
      break;

   case DayOfWeek.Thursday:
      Console.WriteLine("Heute is Donnerstag");
      break;

   case DayOfWeek.Friday:
      Console.WriteLine("Heute is Freitag");
      break;

   case DayOfWeek.Saturday:
      Console.WriteLine("Heute is Samstag");
      break;

   case DayOfWeek.Sunday:
      Console.WriteLine("Heute is Sonntag");
      break;
}
```

Listing 82: Ermitteln des Wochentags eines gegebenen Datums (Forts.)

Wollen Sie einfach nur den Namen des Wochentags ermitteln, finden Sie die Lösung in Rezept 59.

59 Den Namen eines Wochentags ermitteln

Den Namen eines Wochentags können Sie über die Methode `GetDayName` eines `System.Globalization.DateTimeFormatInfo`-Objekts auslesen. Ein solches erhalten Sie über die Eigenschaft `DateTimeFormat` eines `CultureInfo`-Objekts (ebenfalls aus dem Namensraum `System.Globalization`).

Wollen Sie den Namen des Wochentags in der aktuellen Kultur ermitteln, können Sie das dieser Kultur entsprechende `CultureInfo`-Objekt aus der Eigenschaft `CurrentCulture` des aktuellen Thread auslesen, den Sie über `Thread.CurrentThread` (aus dem Namensraum `System.Threading`) erreichen. Ansonsten können Sie einfach über `CultureInfo.CreateSpecificCulture` ein spezifisches `CultureInfo`-Objekt erzeugen und dessen `DateTimeFormat`-Eigenschaft verwenden.

Die Methoden in Listing 83 vereinfachen das Auslesen des Wochentagnamens. Die erste Variante erwartet lediglich einen Wert der Aufzählung `DayOfWeek` und gibt den Namen des Wochentags in der aktuellen Kultur zurück. Die zweite Variante erwartet neben dem Wochentag ein `CultureInfo`-Objekt, das die Kultur verwaltet, für die der Wochentagname zurückgegeben werden soll. Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System`, `System.Globalization` und `System.Threading` einbinden.

```
/* Liefert den Namen des angegebenen Wochentags zurück */
public static string GetDayName(DayOfWeek dayOfWeek)
{
   return Thread.CurrentThread.CurrentCulture.
      DateTimeFormat.GetDayName(dayOfWeek);
}

/* Liefert den Namen des angegebenen Wochentags in der angegebenen
   Kultur zurück */
public static string GetDayName(DayOfWeek dayOfWeek, CultureInfo cultureInfo)
{
   return cultureInfo.DateTimeFormat.GetDayName(dayOfWeek);
}
```

Listing 83: Methoden zum Ermitteln des Namens eines Wochentags in der aktuellen und in einer anzugebenden Kultur

Listing 84 zeigt eine beispielhafte Anwendung dieser Methoden in einer Konsolenanwendung.

```
// Den Namen des Wochentags in der aktuellen Kultur ermitteln
DateTime date = DateTime.Now;
string weekdayName = GetDayName(date.DayOfWeek);
Console.WriteLine("Heute ist " + weekdayName);

// Den Namen des Wochentags in der englischen Kultur ermitteln
date = DateTime.Now;
weekdayName = GetDayName(date.DayOfWeek,
   CultureInfo.CreateSpecificCulture("en"));
Console.WriteLine("Today is " + weekdayName);

// Den Namen des Wochentags in der spanischen Kultur ermitteln
date = DateTime.Now;
weekdayName = GetDayName(date.DayOfWeek,
   CultureInfo.CreateSpecificCulture("es"));
Console.WriteLine("Hoy es " + weekdayName);
```

Listing 84: Beispielhafte Anwendung der GetDayName-Methoden

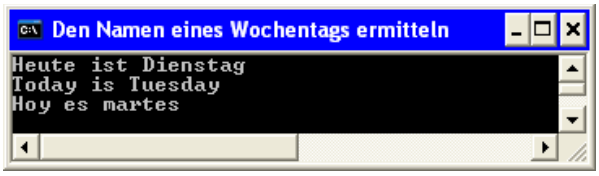


Abbildung 45: Das Beispielprogramm in Aktion

60 Den Namen eines Monats ermitteln

Den Namen eines Monats erhalten Sie auf eine ähnliche Weise wie den Namen eines Wochentags, wie ich es in Rezept 59 beschrieben habe. Der einzige Unterschied ist, dass Sie nun die Methode `GetMonthName` der `DateTimeFormatInfo`-Instanz aufrufen.

Die Methoden `GetMonthName` in Listing 85 helfen Ihnen dabei, den Namen eines Monats zu ermitteln. Die erste Variante, die den Namen für die aktuelle Kultur zurückgibt, ruft die zweite Variante auf, wobei die aktuelle Kultur aus `System.Threading.Thread.CurrentThread.CurrentCulture` ausgelesen wird. Die zweite Variante überprüft zusätzlich den übergebenen Monat und wirft eine `ArgumentException`, falls dieser ungültig ist.

Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System`, `System.Globalization` und `System.Threading` einbinden.

```
/* Liefert den Namen des angegebenen Monats in der aktuellen Kultur */
public static string GetMonthName(int month)
{
   return GetMonthName(month,
      Thread.CurrentThread.CurrentCulture);
}

/* Liefert den Namen des angegebenen Monats in der angegebenen Kultur */
public static string GetMonthName(int month, CultureInfo cultureInfo)
{
   // Den übergebenen Monat überprüfen
   if (month >= 1 && month <= 12)
   {
      return cultureInfo.DateTimeFormat.GetMonthName(month);
   }
   else
   {
      throw new ArgumentException("Der Monat muss ein Wert " +
         "Zwischen 1 und 12 sein");
   }
}
```

Listing 85: Methoden zur Ermittlung des Namens eines Monats

Listing 86 zeigt, wie Sie diese Methoden anwenden.

```
// Den Namen des Monats in der aktuellen Kultur ermitteln
DateTime date = DateTime.Now;
string monthName = GetMonthName(date.Month);
Console.WriteLine("Es ist " + monthName);
```

Listing 86: Beispielhafte Anwendung der GetMonthName-Methoden

```
// Den Namen des Monats in der englischen Kultur ermitteln
date = DateTime.Now;
monthName = GetMonthName(date.Month,
   CultureInfo.CreateSpecificCulture("en"));
Console.WriteLine("It is " + monthName);

// Den Namen des Monats in der spanischen Kultur ermitteln
date = DateTime.Now;
monthName = GetMonthName(date.Month,
   CultureInfo.CreateSpecificCulture("es"));
Console.WriteLine("Es " + monthName);
```

Listing 86: Beispielhafte Anwendung der GetMonthName-Methoden (Forts.)

61 Den ersten und letzten Tag der Woche eines Datums berechnen

Den ersten Tag der Woche, in die ein bestimmtes Datum fällt, können Sie berechnen, indem Sie zunächst einen `int`-Wert ermitteln, der für den im Datum angegebenen Wochentag steht. Dazu fragen Sie die `DayOfWeek`-Eigenschaft ab, die einen Wert der `DayOfWeek`-Aufzählung zurückgibt. Da diese mit der Konstanten `Sunday` beginnt (die den Wert 0 besitzt), müssen Sie den ermittelten Wert so umrechnen, dass für Montage der Wert 1 und für Sonntage der Wert 7 resultiert. Dazu können Sie einfach abfragen, ob der nach `int` konvertierte Wert gleich 0 ist, und setzen diesen dann auf 7. Mit diesem Wert können Sie die Differenz zwischen dem Wochenstart (also dem Montag) und dem Wochentag des Datums berechnen. Diese Differenz ziehen Sie dann einfach von dem Datum ab um das Datum zu erhalten, das dem Montag dieser Woche entspricht. Den letzten Tag der Woche berechnen Sie nun, indem Sie dem Wochen-Startdatum sechs Tage hinzuaddieren:

```
// Datum festlegen
DateTime date = DateTime.Now;

// Einen int-Wert für den Wochentag des Datums ermitteln
int dayOfWeek = (int)date.DayOfWeek;
if (dayOfWeek == 0)
{
   dayOfWeek = 7; // Sonntag
}

// Die Differenz an Tagen zwischen dem Wochenstart (Montag)
// und dem Wochentag des Datums berechnen
int daysFromMonday = dayOfWeek - 1;
DateTime firstDay = date.AddDays(daysFromMonday * -1);
```

Listing 87: Berechnung des Datums des ersten und des letzten Tags der Woche, in die ein gegebenes Datum fällt

```
// Den letzten Tag der Woche ermitteln
DateTime lastDay = firstDay.AddDays(6);
```

Listing 87: Berechnung des Datums des ersten und des letzten Tags der Woche, in die ein gegebenes Datum fällt (Forts.)

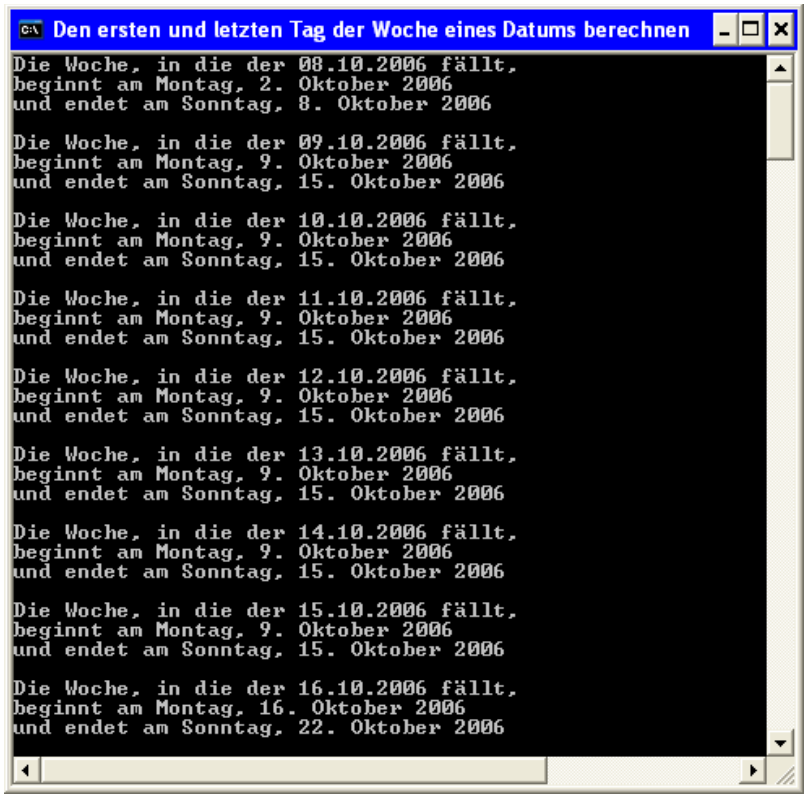


Abbildung 46: Ein Beispielprogramm hat den Start und das Ende der Woche einiger Datumswerte berechnet

62 Den ersten und den letzten Tag eines Monats berechnen

Den ersten Tag eines Monats zu berechnen ist trivial. Dazu müssen Sie lediglich ein neues `DateTime`-Objekt erzeugen, dessen Konstruktor Sie das Jahr, den Monat und für den Tag die Zahl 1 übergeben. Für den letzten Tag können Sie ein `DateTime`-Objekt erzeugen, das mit dem Jahr und dem Monat initialisiert wird, die dem aktuellen Monat addiert mit 1 entsprechen. Als Tag übergeben Sie wieder 1. Dabei müssen Sie natürlich beachten, dass nach dem Monat 12 ein neues Jahr beginnt. Damit erhalten Sie den ersten Tag des folgenden Monats. Über die `AddDays`-Methode ziehen Sie dann einfach einen Tag ab:

```
int month = 12;
int year = 2006;

// Den ersten Tag des Monats ermitteln
DateTime startDate = new DateTime(year, month, 1);

// Den letzten Tag des Monats ermitteln
if (month == 12)
{
   month = 1;
   year++;
}
else
{
   month++;
}
DateTime endDate = new DateTime(year, month, 1).AddDays(-1);
```

Listing 88: Berechnung des ersten und letzten Tags eines Monats

Die Klasse `MonthInfo` in Listing 89, die Sie auch im Repository und in der Codebook-Klassenbibliothek finden, erleichtert die Berechnung des Start- und des Enddatums eines Monats. Beim Erzeugen einer Instanz dieser Klasse müssen Sie lediglich den Monat und das Jahr im Konstruktor übergeben. Danach können Sie die benötigten Datumswerte aus den Eigenschaften `StartDate` und `EndDate` auslesen.

```
public class MonthInfo
{
   private int month;
   /* Das Quartal */
   public int Month
   {
      get { return this.month; }
   }

   private int year;
   /* Das Jahr */
   public int Year
   {
      get { return this.year; }
   }

   private DateTime startDate;
   /* Das Startdatum des Monats */
   public DateTime StartDate
```

Listing 89: Klasse zur Berechnung des Start- und des Enddatums eines Monats

```
   {
      get { return this.startDate; }
   }

   private DateTime endDate;
   /* Das Enddatum des Monats */
   public DateTime EndDate
   {
      get { return this.endDate; }
   }

   /* Konstruktor. Berechnet das Start- und das */
   /* Enddatum des übergebenen Monats. */
   public MonthInfo(int month, int year)
   {
      // Die Argumente überprüfen
      if (month < 1 || month > 12 || year < 0 || year > 9999)
      {
         throw new ArgumentException(
            "Der Monat muss ein Wert zwischen 1 und 12 " +
            "und das Jahr muss ein Wert zwischen 0 und 9999 sein");
      }

      // Argumente übergeben
      this.month = month;
      this.year = year;

      // Den ersten Tag des Monats ermitteln
      this.startDate = new DateTime(year, month, 1);

      // Den letzten Tag des Monats ermitteln
      if (month == 12)
      {
         month = 1;
         year++;
      }
      else
      {
         month++;
      }
      this.endDate = new DateTime(year, month, 1).AddDays(-1);
   }
}
```

Listing 89: Klasse zur Berechnung des Start- und des Enddatums eines Monats (Forts.)

Zum Kompilieren dieser Klasse müssen Sie lediglich den Namensraum `System` einbinden.

Abbildung 47 zeigt eine einfache Anwendung, die über die `MonthInfo`-Klasse das Start- und das Enddatum der Monate von 2006 berechnet hat.

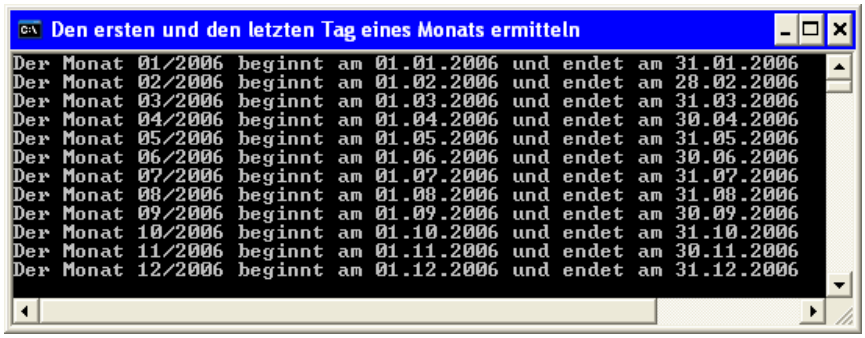


Abbildung 47: Ein Beispielprogramm hat das Start- und das Enddatum der Monate von 2006 berechnet

63 Das Quartal eines gegebenen Datums berechnen

Das Quartal eines gegebenen Datums können Sie recht einfach berechnen. Ich nutze dazu eine Ganzzahl-Division. Der Monat des Datums minus eins wird dabei durch drei (der Anzahl der Monate innerhalb eines Quartals) geteilt. Ist der Monat zum Beispiel 1, 2 oder 3, resultiert 0. Ist der Monat 4, 5 oder 6, resultiert 1 usw. Addiert mit eins ergibt diese Berechnung das aktuelle Quartal.

Zum Kompilieren der Methode in Listing 90 müssen Sie den Namensraum `System` importieren.

```
public static int GetQuarter(this DateTime date)
{
   return ((date.Month - 1) / 3) + 1;
}
```

Listing 90: Methode zur Berechnung des Quartals eines Datums

64 Den ersten und den letzten Tag eines Quartals berechnen

Zur Berechnung des ersten Tags eines angegebenen Quartals können Sie eine neue `DateTime`-Instanz mit dem Jahr des Datums, dem ersten Monat des Quartals und den Tag 1 erzeugen. Das einzige Problem ist die Berechnung des ersten Monats des Quartals. Diesen können Sie berechnen, indem Sie das Quartal mit 3 multiplizieren und 2 vom Ergebnis abziehen.

Den letzten Tag des Quartals können Sie dann berechnen, indem Sie dem Datum des ersten Tages drei Monate aufrechnen und von diesem Datum einen Tag abziehen:

```
// Angabe des Quartals und des Jahrs
int quarter = 1;
int year = 2006

// Ersten Tag im Quartal berechnen
DateTime firstQuarterDay = new DateTime(year, (quarter * 3) - 2, 1);

// Letzten Tag im Quartal berechnen
DateTime lastQuarterDay = firstDay.AddMonths(3).AddDays(-1);
```

Listing 91: Berechnung des Start- und des Enddatums eines gegebenen Quartals

Dieses Beispiel benötigt den Import des Namensraums `System`.

In Listing 92 finden Sie die Klasse `QuarterInfo` zu Berechnung des Start- und des Enddatums, die Sie auch im Repository und in der Codebook-Klassenbibliothek finden. Diese Klasse erleichtert die Berechnung, indem Sie dem Konstruktor lediglich das Quartal und das Jahr übergeben und die berechneten Datumswerte aus den Eigenschaften `StartDate` und `EndDate` auslesen können. Zum Kompilieren dieser Klasse müssen Sie den Namensraum `System` einbinden.

```
public class QuarterInfo
{
   private int quarter;
   /* Das Quartal */
   public int Quarter
   {
      get { return this.quarter; }
   }

   private int year;
   /* Das Jahr */
   public int Year
   {
      get { return this.year; }
   }

   private DateTime startDate;
   /* Das Startdatum des Quartals */
   public DateTime StartDate
   {
      get { return this.startDate; }
   }

   private DateTime endDate;
```

Listing 92: Klasse zur Berechnung des Start- und des Enddatums eines gegebenen Quartals

```
   /* Das Enddatum des Quartals */
   public DateTime EndDate
   {
      get { return this.endDate; }
   }

   /* Konstruktor. Berechnet das Start- und das
      Enddatum des übergebenen Quartals. */
   public QuarterInfo(int quarter, int year)
   {
      // Das übergebene Quartal und das Jahr überprüfen
      if (quarter >= 1 && quarter <= 4 &&
         year >= 0 && year <= 9999)
      {
         // Quartal und Jahr übergeben
         this.quarter = quarter;
         this.year = year;

         // Ersten Tag im Quartal berechnen
         this.startDate = new DateTime(year, (quarter * 3) - 2, 1);

         // Letzten Tag im Quartal berechnen
         this.endDate = this.startDate.AddMonths(3).AddDays(-1);
      }
      else
      {
         throw new ArgumentException("Das Quartal muss eine Zahl " +
            "zwischen 1 und 4 und das Jahr eine Zahl zwischen 0 " +
            "und 9999 sein");
      }
   }
}
```

Listing 92: Klasse zur Berechnung des Start- und des Enddatums eines gegebenen Quartals (Forts.)

Abbildung 48 zeigt die Ausgabe einer Beispielanwendung, die die Quartale der Jahre von 2006 bis 2010 berechnet hat.

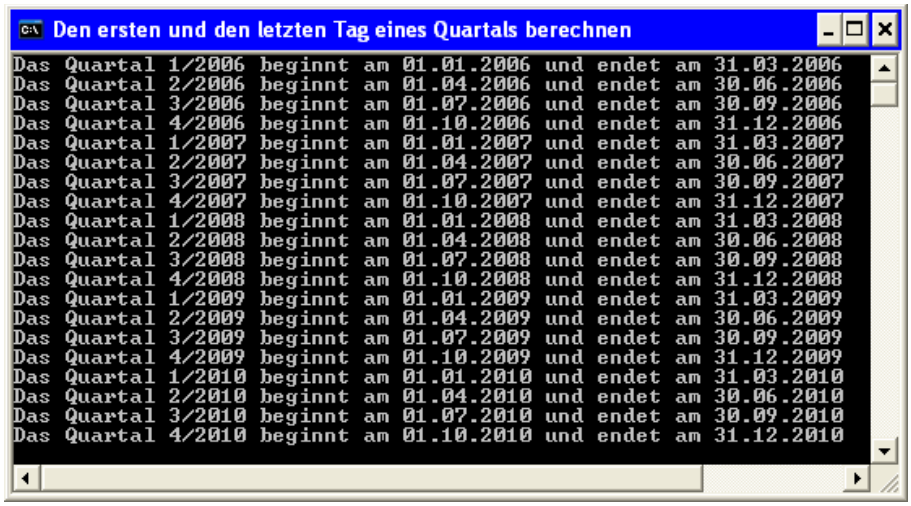


Abbildung 48: Ein Beispielprogramm hat das Start- und das Enddatum der Quartale der Jahre 2006 bis 2010 berechnet

65 Ermitteln, ob ein Jahr ein Schaltjahr ist

Über die statische Methode `IsLeapYear` der `DateTime`-Struktur können Sie ermitteln, ob ein Jahr ein Schaltjahr ist. `IsLeapYear` gibt `true` zurück, wenn es sich um ein Schaltjahr handelt.

Listing 93 ermittelt, ob das aktuelle Jahr ein Schaltjahr ist. Das Beispiel erfordert den Import des Namensraums `System`.

```
// Aktuelles Datum ermitteln
DateTime date = DateTime.Now;

// Ermitteln, ob es sich um ein Schaltjahr handelt
if (DateTime.IsLeapYear(date.Year))
{
   Console.WriteLine("{0} ist ein Schaltjahr", date.Year);
}
else
{
   Console.WriteLine("{0} ist kein Schaltjahr", date.Year);
}
```

Listing 93: Ermitteln, ob das aktuelle Jahr ein Schaltjahr ist

66 Ermitteln der ISO-Kalenderwoche

Die ISO[1]-Norm 8601, nach der in Deutschland die Kalenderwoche berechnet wird, definiert u. a. die erste Woche im Jahr als die Woche, in die mindestens vier der ersten sieben Januartage fallen (also die Woche, die den ersten Donnerstag im Jahr enthält). Danach ergeben sich die folgenden Aussagen:

- Beginnt das Jahr mit einem Montag, so entspricht der 1.1. der ersten Kalenderwoche und das Vorjahr schließt mit dem Ende der letzten Kalenderwoche ab (logisch ☺).
- Beginnt das Jahr mit einem Dienstag, Mittwoch oder Donnerstag, werden die ersten Tage der Woche, die im Vorjahr liegen, der ersten Kalenderwoche des neuen Jahres zugerechnet.
- Beginnt das Jahr mit einem Freitag, Samstag oder Sonntag, werden diese Tage der letzten Kalenderwoche des Vorjahres zugerechnet.
- Je nachdem, wann die erste Woche im Jahr startet und die letzte endet, kann das Jahr 52 oder 53 Wochen umfassen.

Die Kalenderwoche eines Datums können Sie (mit einem kleinen Fehler) über eine Instanz der von `Calendar` abgeleiteten Klasse ermitteln, die den aktuellen Kalender des Systems darstellt. Diese Instanz ermitteln Sie über die `Calendar`-Eigenschaft der Klasse `System.Globalization.CultureInfo.CurrentCulture`. Über die Methode `GetWeekOfYear` berechnen Sie die Kalenderwoche eines Datums, das Sie am ersten Argument übergeben. Am zweiten Argument definieren Sie mit einem Wert der Aufzählung `CalendarWeekRule`, welche Woche als erste Kalenderwoche gezählt wird. Dazu stehen Ihnen die Werte der `CalendarWeekRule`-Aufzählung zur Verfügung. Der Wert `FirstDay` bestimmt, dass die Woche mit dem ersten Januar die erste Woche ist, `FirstFourDayWeek`, dass die Woche die erste ist, die mindestens vier Tage im neuen Jahr besitzt (Standard für Deutschland), über den Wert `FirstFullWeek` legen Sie die erste im neuen Jahr komplette Woche als Woche 1 fest. Am letzten Argument spezifizieren Sie mit den Werten der `DayOfWeek`-Aufzählung (deren Konstanten die Wochentagsnamen sind), mit welchem Tag die Woche beginnt (Sonntag in den USA, Montag in Deutschland). An den beiden letzten Argumenten sollten Sie die aktuellen Kultureinstellungen übergeben, die Sie aus der Eigenschaft `DateTimeFormat` eines `CultureInfo`-Objekts auslesen können.

Listing 94 implementiert dies in Form der Methode `GetCalendarWeek`, die die Kalenderwoche eines übergebenen Datums ermittelt und in einer Instanz der Klasse `CalendarWeek` zurückgibt.

Hinweis

Die `GetWeekOfYear`-Methode eines `Calendar`-Objekts berechnet die Kalenderwoche leider nicht genau nach dem ISO-8601-Standard. Am Ende des Jahres treten häufig Fehler auf. Für den 31.12.2007 (der ein Montag ist) ergibt die Berechnung z. B. 53, obwohl diese Woche sechs Tage (Dienstag bis Sonntag) im Jahr 2008 besitzt und deshalb eigentlich die Woche 1 sein sollte. Die Berechnung für den 1.1.2008 ergibt dann auch wie erwartet 1. Microsoft dokumentiert dieses Verhalten im Knowledge-Base-Artikel 200299 (für Visual Basic 6). Um diesen »Bug« zu umschiffen, überprüft `GetCalendarWeek`, ob die ermittelte Kalenderwoche größer als 52 ist und ob die für das Datum addiert mit

1. ISO (International Organization for Standardization) ist ein Schweizer Institut, das internationale Standards entwickelt und veröffentlicht.

einer Woche ermittelte Kalenderwoche 2 ergibt. In diesem Fall wird die Kalenderwoche 1 zurückgegeben.

Dieser kleine Bugfix hat sich als zuverlässig erwiesen. In einem Test wurde die Kalenderwoche für die Jahre 1899 bis 2999 korrekt berechnet (was ich über die weiter unten beschriebene separate Methode zur Berechnung der deutschen Kalenderwoche und teilweise über den Kalender von Outlook überprüft habe).

`GetCalendarWeek` gibt neben der Kalenderwoche auch das Jahr zurück, da es vorkommen kann, dass ein Datum in der letzten Woche eines Jahres der ersten Kalenderwoche des nächsten Jahres oder ein Datum in der ersten Woche eines Jahres der letzten Kalenderwoche des Vorjahres zugeordnet wird. Das Jahr wird deshalb zunächst aus dem übergebenen Datum ausgelesen. Dann überprüft `GetCalendarWeek`, ob die Woche 1 ermittelt wurde und der Monat des Datums 12 ist. In diesem Fall wurde das Datum der ersten Kalenderwoche des neuen Jahres zugeordnet und das Jahr wird folglich um 1 erhöht. Dann wird noch überprüft, ob die Kalenderwoche 52 oder 53 und der Monat des Datums 1 ist. In diesem Fall wurde das Datum der letzten Kalenderwoche des Vorjahres zugeordnet und das Jahr wird um 1 erniedrigt.

Mit den ermittelten Daten erzeugt `GetCalendarWeek` schließlich ein `CalendarWeek`-Objekt (dessen Klasse in Listing 94 deklariert ist) und gibt eine Referenz darauf zurück.

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
/* Verwaltet die Daten einer Kalenderwoche */
public class CalendarWeek
{
   public int Year;
   public int Week;

   public CalendarWeek(int year, int week)
   {
      this.Year = year;
      this.Week = week;
   }
}

/* Berechnet die Kalenderwoche eines internationalen Datums.
 * Diese Methode berechnet die Kalenderwoche eines Datums
 * nach der GetWeekOfYear-Methode eines Calendar-Objekts
 * und korrigiert den darin enthaltenen Fehler. */
public static CalendarWeek GetCalendarWeek(this DateTime date)
{
   // Aktuelle Kultur ermitteln
   CultureInfo currentCulture = CultureInfo.CurrentCulture;
```

Listing 94: Methode zur Ermittlung der Kalenderwoche eines Datums über ein Calendar-Objekt

```
   // Aktuellen Kalender ermitteln
   Calendar calendar = currentCulture.Calendar;

   // Kalenderwoche über das Calendar-Objekt ermitteln
   int calendarWeek = calendar.GetWeekOfYear(date,
      currentCulture.DateTimeFormat.CalendarWeekRule,
      currentCulture.DateTimeFormat.FirstDayOfWeek);

   // Überprüfen, ob eine Kalenderwoche größer als 52
   // ermittelt wurde und ob die Kalenderwoche des Datums
   // in einer Woche 2 ergibt: In diesem Fall hat
   // GetWeekOfYear die Kalenderwoche nicht nach ISO 8601
   // berechnet (Montag, der 31.12.2007 wird z.B.
   // fälschlicherweise als KW 53 berechnet).
   // Die Kalenderwoche wird dann auf 1 gesetzt
   if (calendarWeek > 52)
   {
      date = date.AddDays(7);
      int testCalendarWeek = calendar.GetWeekOfYear(date,
         currentCulture.DateTimeFormat.CalendarWeekRule,
         currentCulture.DateTimeFormat.FirstDayOfWeek);
      if (testCalendarWeek == 2)
      {
         calendarWeek = 1;
      }
   }

   // Das Jahr der Kalenderwoche ermitteln
   int year = date.Year;
   if (calendarWeek == 1 && date.Month == 12)
   {
      year++;
   }
   if (calendarWeek >= 52 && date.Month == 1)
   {
      year--;
   }

   // Die ermittelte Kalenderwoche zurückgeben
   return new CalendarWeek(year, calendarWeek);
}
```

Listing 94: Methode zur Ermittlung der Kalenderwoche eines Datums über ein Calendar-Objekt (Forts.)

Eine andere Möglichkeit, die Kalenderwoche zu berechnen, ist ein eigener Algorithmus. Im Internet kursieren verschiedene Algorithmen, die aber teilweise Kalenderwochen falsch berechnen. Ich habe selbst versucht, eine Berechnungs-Methode zu entwickeln, bin aber an

einigen wenigen Sonderfällen gescheitert. Die Berechnung ist nicht trivial, auch wenn es auf den ersten Blick so erscheint.

Der Algorithmus, auf dem die Methode `GetGermanCalendarWeek` in Listing 95 basiert und den ich in einem Newsgroup-Beitrag gefunden habe, berechnet die Kalenderwoche anscheinend (und nach meinen Tests) sehr genau. Ein Vergleich mit der Methode `GetCalendarWeek` hat auf jeden Fall ergeben, dass beide Methoden die Kalenderwochen der Jahre 1899 bis 2999 gleich berechnen. Ich habe den Algorithmus nahezu unverändert übernommen, aber um die Berechnung des Jahres für die Kalenderwoche erweitert. Das Jahr wird auf dieselbe Weise berechnet wie in der Methode `GetCalendarWeek`.

`GetGermanCalendarWeek` erfordert den Import des Namensraums `System`.

```
/* Methode zur Berechnung einer deutschen Kalenderwoche nach dem
 * C++-Algorithmus von Ekkehard Hess aus einem Beitrag vom 29.7.1999
 * in der Newsgroup borland.public.cppbuilder.language
 * (freigegeben zur allgemeinen Verwendung) */
public static CalendarWeek GetGermanCalendarWeek(this DateTime date)
{
   double a = Math.Floor((14 - (date.Month)) / 12D);
   double y = date.Year + 4800 - a;
   double m = (date.Month) + (12 * a) - 3;

   double jd = date.Day + Math.Floor(((153 * m) + 2) / 5) +
      (365 * y) + Math.Floor(y / 4) - Math.Floor(y / 100) +
      Math.Floor(y / 400) - 32045;

   double d4 = (jd + 31741 - (jd % 7)) % 146097 % 36524 %
      1461;
   double L = Math.Floor(d4 / 1460);
   double d1 = ((d4 - L) % 365) + L;

   // Kalenderwoche ermitteln
   int calendarWeek = (int) Math.Floor(d1 / 7) + 1;

   // Das Jahr der Kalenderwoche ermitteln
   int year = date.Year;
   if (calendarWeek == 1 && date.Month == 12)
      year++;
   if (calendarWeek >= 52 && date.Month == 1)
      year--;

   // Die ermittelte Kalenderwoche zurückgeben
   return new CalendarWeek(year, calendarWeek);
}
```

Listing 95: Methode zur Ermittlung der Kalenderwoche eines deutschen Datums über einen komplexen Algorithmus

67 Ermitteln, mit welchem Datum eine Kalenderwoche beginnt

Startdatum einer deutschen Kalenderwoche ermitteln

Das Datum des Montags einer beliebigen Kalenderwoche können Sie für Deutschland eigentlich recht einfach berechnen. Der DIN-Norm für Datumsberechnungen (DIN 1355) können Sie entnehmen, dass der 4.1. eines Jahres immer in die Woche 1 fällt (Dank an die Kollegen, die das Access-Codebook geschrieben haben ☺). Mit diesem Datum können Sie das Datum ermitteln, das den Montag dieser Woche repräsentiert. Um das Startdatum einer Kalenderwoche zu ermitteln, können Sie dem Startdatum der ersten Kalenderwoche ganz einfach die Anzahl der Kalenderwochen -1 multipliziert mit 7 addieren, wie es Listing 96 zeigt. Der Montag der Woche, in die der 4.1. fällt, wird in diesem Beispiel über eine Ermittlung der Wochentagszahl berechnet. Die `DayOfWeek`-Aufzählung eines `DateTime`-Objekts verwaltet den Wochentag als Zahl von 0 (Sonntag) bis 6 (Samstag). Da die 0 den Sonntag darstellt, muss dieser extra behandelt werden. Ansonsten wird der Montag einfach berechnet, indem die Wochentagnummer -1 vom Datum abgezogen wird. Mit diesem Basisdatum wird dann einfach über eine Addition der Tage der Kalenderwochen bis zur abgefragten Woche das Startdatum dieser Woche ermittelt.

Die Methode `GetGermanCalendarWeekStartDate` erfordert die Einbindung der Namensräume `System` und `System.Globalization`.

```
/* Berechnet das Startdatum einer deutschen Kalenderwoche */
public static System.DateTime GetGermanCalendarWeekStartDate(
   int calendarWeek, int year)
{
   // Datum für den 4.1. des Jahres ermitteln
   System.DateTime baseDate = new System.DateTime(year, 1, 4);

   // Den Montag dieser Woche ermitteln
   int dayOfWeek = (int)baseDate.DayOfWeek;
   if (dayOfWeek > 0)
   {
      // Montag bis Samstag
      baseDate = baseDate.AddDays((dayOfWeek - 1) * -1);
   }
   else
   {
      // Sonntag
      baseDate = baseDate.AddDays(-6);
   }

   // Das Ergebnisdatum ermitteln
   return baseDate.AddDays((calendarWeek - 1) * 7);
}
```

Listing 96: Ermitteln des Startdatums einer deutschen Kalenderwoche

Startdatum einer internationalen Kalenderwoche ermitteln

Die Methode `GetGermanCalendarWeekStartDate` ist nicht geeignet für Länder, bei denen die erste Kalenderwoche anders gerechnet wird als in Deutschland und/oder bei denen die Woche mit einem anderen Tag als dem Montag beginnt. Die USA, wo die Woche mit dem Sonntag beginnt, ist ein Beispiel dafür. Um auch für diese Länder das Startdatum einer Kalenderwoche berechnen zu können, habe ich die (komplexere und deutlich langsamere) Methode `GetCalendarWeekStartDate` in Listing 97 entwickelt. Diese Methode ermittelt zunächst das Startdatum des Jahres (der 1.1. des übergebenen Jahres). Ausgehend von diesem Datum, das ja z. B. ein Mittwoch sein kann, wird dann das Datum des ersten Wochentags dieser Woche ermittelt. Dazu überprüft das Programm in einer Schleife, ob der Wochentag des Basisdatums dem aktuell eingestellten Wochenstart-Wochentag entspricht (der aus der Eigenschaft `CurrentCulture.DateTimeFormat.FirstDayOfWeek` der `CultureInfo`-Klasse aus dem Namensraum `System.Globalization` ausgelesen werden kann) und reduziert das Basisdatum in der Schleife jeweils um einen Tag (eine einfachere Lösung habe ich nicht gefunden, da die Wochen international ja mit unterschiedlichen Tagen beginnen können).

Mit dem ermittelten Basisdatum wird dann über die Methode `GetGermanCalendarWeek` aus dem Rezept 66 die Kalenderwoche berechnet. Handelt es sich nicht um die Woche 1, addiert `GetCalendarWeekStartDate` einfach sieben Tage auf das Basisdatum, um den Wochenbeginn der ersten Kalenderwoche zu finden. Auf das so ermittelte Datum werden dann noch die Tage bis zur angegebenen Kalenderwoche addiert.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.Globalization` importieren.

```
/* Die Implementierung der Klasse CalendarWeek und der Methoden
 * GetCalendarWeek und GetGermanCalendarWeek aus dem vorhergehenden Rezept
 * wird aus Übersichtsgründen hier nicht dargestellt */

/* Berechnet das Startdatum einer internationalen Kalenderwoche */
public static System.DateTime GetCalendarWeekStartDate(
   int calendarWeek, int year)
{
   // Basisdatum (1.1. des angegebenen Jahres) ermitteln
   System.DateTime startDate = new System.DateTime(
      year, 1, 1);

   // Das Datum des ersten Wochentags dieser Woche ermitteln
   while (startDate.DayOfWeek !=
      CultureInfo.CurrentCulture.DateTimeFormat.FirstDayOfWeek)
      startDate = startDate.AddDays(-1);
```

Listing 97: Berechnen des Startdatums einer Kalenderwoche

```
   // Die Kalenderwoche ermitteln: Wenn es sich um die Woche 1 handelt,
   // ist dies das Basisdatum für die Berechnung, wenn nicht, müssen
   // sieben Tage aufaddiert werden
   CalendarWeek cw = startDate.GetCalendarWeek();
   if (cw.Week != 1)
      startDate = startDate.AddDays(7);

   // Das Ergebnisdatum ermitteln
   return startDate.AddDays((calendarWeek - 1) * 7);
}
```

Listing 97: Berechnen des Startdatums einer Kalenderwoche (Forts.)

Hinweis

Die Berechnung des Startdatums ist aufgrund eines unerwarteten Nebeneffekts sehr tolerant. So können Sie als Kalenderwoche auch 53 übergeben, wenn das Jahr nur 52 Wochen besitzt. `GetCalendarWeekStartDate` sieht diese Woche dann einfach als Woche 1 des Folgejahres an.

Um die Berechnung der Kalenderwoche zu testen, habe ich (neben dem Unit-Test, der in die Codebook-Klassenbibliotheks-Solution eingebunden ist) im Beispiel-Projekt die Startdaten aller Kalenderwochen der Jahre 2008 und 2009 berechnet:

```
// Startdatums-Werte für die Kalenderwochen zwischen 2008
// und 2009 berechnen
for (int year = 2008; year <= 2009; year++)
{
   for (int calendarWeek = 1; calendarWeek < 54;
      calendarWeek++)
   {
      // Das Startdatum dieser Kalenderwoche ermitteln
      // und ausgeben
      DateTime date =
         GetGermanCalendarWeekStartDate(
         calendarWeek, year);

      Console.WriteLine("KW {0:00}/{1}: {2}", calendarWeek,
         year, date.ToShortDateString());
   }
}
```

Listing 98: Testweises Berechnen der Startdatums-Werte der Kalenderwochen der Jahre 2008 und 2009

Abbildung 49 und eine stichprobenartige Überprüfung der erzeugten Daten beweisen, dass die Berechnung in Ordnung ist ☺.

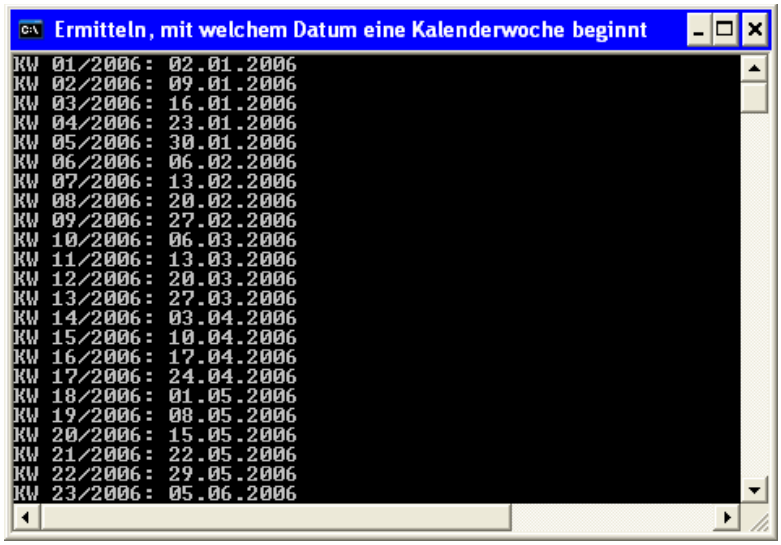


Abbildung 49: Die vom Testprogramm erzeugte Textdatei in einer verkürzten Fassung

68 Anzahl der Kalenderwochen eines Jahres bestimmen

Die Anzahl der Kalenderwochen eines Jahres können Sie über die Methode `GetGermanCalendarWeek` aus dem Rezept 66 ganz einfach bestimmen, indem Sie die Kalenderwoche des 31.12. des Jahres ermitteln. Handelt es sich um die Woche 1, ermitteln Sie die Kalenderwoche des Datums, das eine Woche zurückliegt. Die Methode `GetCalendarWeekCount` in Listing 99 nimmt Ihnen diese Arbeit ab. Diese Methode erfordert das Vorhandensein der Methode `GetCalendarWeek` (und damit auch der Klasse `CalendarWeek`) aus dem Rezept 66 und den Import des Namensraums `System`.

```
public static int GetCalendarWeekCount(int year)
{
   // Kalenderwoche des 31.12. des Jahres ermitteln
   System.DateTime baseDate = new System.DateTime(year, 12, 31);
   CalendarWeek calendarWeek = baseDate.GetCalendarWeek();

   // Wenn dieser Tag in die Woche 1 des neuen Jahres fällt, die Kalenderwoche
   // des um eine Woche reduzierten Datums ermitteln
   if (calendarWeek.Week == 1)
      return baseDate.AddDays(-7).GetCalendarWeek().Week;

   // Ergebnis zurückgeben
   return calendarWeek.Week;
}
```

Listing 99: Methode zur Ermittlung der Kalenderwochen-Anzahl eines Jahres

69 Systemdatum und -zeit setzen

Das Setzen des Systemdatums und der Systemzeit ist scheinbar nicht mit .NET-Klassen möglich. Die entsprechenden Eigenschaften der `DateTime`-Struktur sind schreibgeschützt und `DateTime` besitzt keine Methoden zum Schreiben der Systemdatumswerte.

Das Systemdatum und die Systemzeit können Sie aber über die API-Funktion `SetSystemTime` setzen. Diese Funktion erwartet am einzigen Argument eine Instanz der API-Struktur `SYSTEMTIME`, die einzelne Felder für die Bestandteile eines Datums besitzt. Diese Struktur füllen Sie mit den gewünschten Werten und rufen damit `SetSystemTime` auf. Dabei müssen Sie beachten, dass diese Funktion die Zeitangabe im UTC-Format (Coordinated Universal Time) erwartet. Diese Zeitangabe, die früher als *Greenwich Mean Time (GMT)* bezeichnet wurde, bezieht sich auf die aktuelle Zeit in der englischen Ortschaft Greenwich und wird als internationale Basis für Zeitangaben verwendet. Die Zeit in Deutschland besitzt im Winter einen Offset von einer und im Sommer einen Offset von zwei Stunden zu UTC. Ein Datum können Sie ganz einfach über die Methode `ToUniversalTime` eines `DateTime`-Objekts in das UTC-Format umrechnen.

Die Methoden in Listing 103 helfen Ihnen dabei, das Systemdatum und die Systemzeit zu setzen. Zunächst habe ich die Struktur `SYSTEMTIME` und die API-Funktion `SetSystemTime` deklariert. `SYSTEMTIME` ist mit dem Attribut `StructLayout` mit dem Argument `LayoutKind.Sequential` versehen, damit die (.NET-)Struktur im Speicher als normale C-Struktur angelegt wird, bei der die einzelnen Elemente (anders als bei .NET) sequenziell hintereinander im Speicher positioniert werden. Die einzelnen Felder sind entsprechend der originalen Deklaration der Struktur (in *winbase.h*) als `short` deklariert. `SetSystemTime` bekommt einen Zeiger auf eine Instanz dieser Struktur übergeben, weswegen das entsprechende Argument bei der Deklaration dieser Funktion in C# mit `ref` deklariert werden muss.

Zum Kompilieren der im Folgenden beschriebenen Methoden müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
/* Struktur für das Setzen der Systemzeit */
[StructLayout(LayoutKind.Sequential)]
private struct SYSTEMTIME
{
   public short uYear;
   public short uMonth;
   public short uDayOfWeek;
   public short uDay;
   public short uHour;
   public short uMinute;
   public short uSecond;
   public short uMilliseconds;
}
```

Listing 100: Deklaration der benötigten API-Struktur und -Funktion

Dann folgt die Deklaration der benötigten API-Funktionen. Neben `SetSystemTime` habe ich auch `FormatMessage` und die für das Lesen einer Systemmeldung verwendete Konstante deklariert um Fehlermeldungen auslesen zu können:

```
[DllImport("kernel32.dll", SetLastError=true)]
static extern int SetSystemTime(ref SYSTEMTIME lpSystemTime);

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
   int dwMessageId, int dwLanguageId, System.Text.StringBuilder lpBuffer,
   int nSize, string [] Arguments);
private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;
```

Listing 101: Deklaration der benötigten API-Funktionen und -Konstanten

Die private Methode `SetSystemTime` in Listing 102 übernimmt den Aufruf der API-Funktion. Tritt beim Aufruf ein Fehler auf, liest die Methode den Fehlercode aus, ermittelt dazu über `FormatMessage` eine passende Fehlerbeschreibung und wirft eine Ausnahme. Ich habe diese Methode implementiert um die aufwändige Fehlerbehandlung nicht in jede der öffentlichen Methoden integrieren zu müssen.

```
private static void SetSystemTime(SYSTEMTIME systemTime)
{
   // Systemzeit setzen
   if (SetSystemTime(ref systemTime) == 0)
   {
      // Fehler bei der Ausführung: Den letzten Windows-Fehlercode
      // auslesen, in eine Beschreibung umwandeln und damit
      // eine Ausnahme werfen
      int apiError = Marshal.GetLastWin32Error();
      StringBuilder errorMessage = new StringBuilder(1024);
      FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, (IntPtr)0, apiError,
         0, errorMessage, 1024, null);
      throw new Exception(errorMessage.ToString());
   }
}
```

Listing 102: Methode zum Setzen der Systemzeit mit Fehlerbehandlung

Die Methode `SetSystemDateTime` in Listing 103 bekommt nun ein `DateTime`-Objekt übergeben und setzt das komplette Datum inklusive der Zeit. Dabei wird das übergebene Datum zunächst in UTC umgerechnet. Dann erzeugt diese Methode eine `SYSTEMTIME`-Struktur, liest die Bestandteile des Datums über die entsprechenden Eigenschaften des `DateTime`-Objekts und schreibt diese in die Struktur. Diese Struktur wird dann an die private Methode `SetSystemTime` übergeben.

Die Methode `SetSystemDate` arbeitet ähnlich, setzt aber nur den Datumsteil des Systemdatums. Der Zeitanteil wird dazu einfach aus dem aktuellen Datum ausgelesen und in die Struktur geschrieben. `SetSystemTime` setzt auf eine ähnliche Weise nur den Zeitanteil. Die übergebene Zeit wird dazu natürlich wieder in eine UTC-Zeit umgerechnet.

```
/* Setzt das Systemdatum inklusive der Zeit */
public static void SetSystemDateTime(DateTime date)
{
   // Zeit in UTC umrechnen
   date = date.ToUniversalTime();

   // SYSTEMTIME-Struktur erzeugen und initialisieren
   SYSTEMTIME systemTime = new SYSTEMTIME();
   systemTime.uYear = (short)date.Year;
   systemTime.uMonth = (short)date.Month;
   systemTime.uDay = (short)date.Day;
   systemTime.uHour = (short)date.Hour;
   systemTime.uMinute = (short)date.Minute;
   systemTime.uSecond = (short)date.Second;
   systemTime.uMilliseconds = (short)date.Millisecond;

   // Systemzeit über die private Methode setzen
   SetSystemTime(systemTime);
}

/* Setzt das Systemdatum ohne Zeit */
public static void SetSystemDate(DateTime date)
{
   // SYSTEMTIME-Struktur erzeugen und initialisieren
   SYSTEMTIME systemTime = new SYSTEMTIME();
   systemTime.uYear = (short)date.Year;
   systemTime.uMonth = (short)date.Month;
   systemTime.uDay = (short)date.Day;
   systemTime.uHour = (short)DateTime.Now.Hour;
   systemTime.uMinute = (short)DateTime.Now.Minute;
   systemTime.uSecond = (short)DateTime.Now.Second;
   systemTime.uMilliseconds = (short)DateTime.Now.Millisecond;

   // Systemzeit über die private Methode setzen
   SetSystemTime(systemTime);
}

/* Setzt die Systemzeit (ohne Datum) */
public static void SetSystemTime(DateTime date)
{
   // Zeit in UTC umrechnen
   date = date.ToUniversalTime();

   // SYSTEMTIME-Struktur erzeugen und initialisieren
   SYSTEMTIME systemTime = new SYSTEMTIME();
```

Listing 103: Methoden zum Setzen des Systemdatums und der Systemzeit

```
    systemTime.uYear = (short)DateTime.Now.Year;
    systemTime.uMonth = (short)DateTime.Now.Month;
    systemTime.uDay = (short)DateTime.Now.Day;
    systemTime.uHour = (short)date.Hour;
    systemTime.uMinute = (short)date.Minute;
    systemTime.uSecond = (short)date.Second;
    systemTime.uMilliseconds = (short)date.Millisecond;

    // Systemzeit über die private Methode setzen
    SetSystemTime(systemTime);
}
```

Listing 103: Methoden zum Setzen des Systemdatums und der Systemzeit (Forts.)

Voraussetzung für die Ausführung der Methoden zum Setzen der Systemzeit ist, laut der Dokumentation der `SetSystemTime`-Funktion, dass das Benutzerkonto, unter dem die Methoden ausgeführt werden, unter Windows NT, 2000 und XP das Privileg `SE_SYSTEMTIME_NAME` besitzt. Dieses Privileg können Sie über die lokalen Sicherheitsrichtlinien in der Systemsteuerung vergeben. Dort wird das Privileg als *Ändern der Systemzeit* bezeichnet. Für den Fall, dass Sie dieses Privileg nicht jedem Benutzer der Anwendung zuordnen können, können Sie die Anwendung auch vor der Ausführung der Methoden zum Setzen der Systemzeit mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 307 beschreibe.

70 Atomuhr-Datum inklusive Uhrzeit von einem NIST-Server abfragen

Das amerikanische Institut NIST (National Institute of Standards and Technology), das hauptsächlich Messverfahren und Standards für die amerikanische Industrie entwickelt (siehe *www.nist.gov*), betreibt einige hochgenaue Atomuhren, deren Datum und Uhrzeit in den USA als Standard gelten. Einige von NIST betriebene TCP/UDP-Server liefern das Datum dieser Uhren als UTC[2]-Zeit. Eine Liste dieser Server finden Sie an der Adresse *www.boulder.nist.gov/timefreq/service/time-servers.html*.

Diese Server implementieren u. a. das »Daytime Protocol« des W3C (RFC 867) über den Port 13 (siehe *www.boulder.nist.gov/timefreq/service/its.htm*). Dieses Protokoll sieht lediglich vor, dass ein Server auf dem Port 13 auf eine Verbindung wartet und das aktuelle Datum inklusive Uhrzeit in einer beliebigen Textform zurücksendet. Die NIST-Server verwenden dazu ein festgelegtes Format in der folgenden Form:

```
JJJJJ YR-MO-DA HH:MM:SS TT L H msADV UTC(NIST) OTM
```

- *JJJJ* ist ein (für uns uninteressantes) modifiziertes Julianisches Datum,
- *YR-MO-DA* ist das normale Datum in der Form *Jahr-Monat-Tag*,
- *HH:MM:SS* ist die Zeit,

2. UTC = Coordinated Universal Time. Eine UTC-Zeitangabe, die früher als *Greenwich Mean Time (GMT)* bezeichnet wurde, bezieht sich auf die aktuelle Zeit in der englischen Ortschaft Greenwich und wird als internationale Basis für Zeitangaben verwendet.

- *TT* sagt aus, ob die USA gerade die Standard- oder die Sommerzeit verwenden,
- *L* sagt mit 0 oder 1 aus, ob im aktuellen Monat am letzten Tag eine extra Sekunde hinzugefügt wird (wegen der Schaltjahres-Berechnung),
- *H (Health)* gibt die »Gesundheit« des Servers an. Bei einer 0 ist der Server in Ordnung und liefert die genaue Zeit, bei 1 kann es sein, dass die Zeit mit einer Ungenauigkeit von fünf Sekunden zurückgegeben wird, weil der Server stark belastet ist, bei 2 ist die Differenz schon zehn Sekunden, bei 4 ist ein schwerer Fehler aufgetreten,
- *msADV* gibt einen Offset in Millisekunden an, den der NIST-Server auf die ermittelte Zeit wegen erwarteter Netzwerk-Zeitverzögerungen aufaddiert (zurzeit 50),
- *UTC(NIST)* gibt an, dass es sich um eine UTC-Zeit von NIST handelt,
- *OTM* ist immer ein Asterisk (*), der beim Empfang verwendet werden kann, um den genauen Zeitpunkt zu bestimmen. Die Zeitangabe im String bezieht sich (mit dem Offset) immer auf den Empfang des OTM-Zeichens.

Über eine Instanz der Klasse `TcpClient` ist es nun relativ einfach, einen NIST-Server abzufragen. Listing 104 setzt dies um. Die Methode `GetNISTTime` fragt dazu gleich mehrere NIST-Server ab, um das genaue Datum auch dann zu erhalten, wenn ein Server nicht erreichbar ist oder einen Gesundheitsstatus größer 0 zurückliefert. Die Abfrage erfolgt in einer Schleife, die alle im anfänglich initialisierten Array `servers` verwalteten Server-Adressen durchgeht. Die NIST-Server, die ich in diesem Array aufgenommen habe, haben sich in meinen Tests als zuverlässig erwiesen. Die Abfrage der Server erfolgt über eine `TcpClient`-Instanz, die natürlich vor der Schleife erzeugt wird. Der Empfangs-Timeout dieser Instanz wird dabei auf eine Sekunde (1.000 Millisekunden) gesetzt, damit die `Read`-Methode nicht unnötig lange auf die Daten wartet. Der standardmäßige Timeout von 0 würde ansonsten bewirken, dass `Read` ganze 20 Sekunden auf Daten wartet (das ist jedenfalls die empirisch ermittelte Standard-Einstellung auf meinem Rechner, das Defaultverhalten ist nicht dokumentiert). Leider besteht nach meinen Recherchen keine direkte Möglichkeit einen Timeout für den Verbindungsaufbau (über die `Connect`-Methode) zu definieren. `Connect` wartet bei einem unerreichbaren Host standardmäßig (nach meinen Versuchen und Informationen aus Newsgroups) zehn Sekunden.

In der Schleife versucht das Programm zunächst über die `Connect`-Methode der `TcpClient`-Instanz zum Port 13 des aktuellen Servers eine Verbindung aufzubauen. Schlägt diese fehl (was über die Ausnahmebehandlung erkannt wird), speichert `GetNISTTime` den aufgetretenen Fehler in der Stringvariable `errors`. Die Schleife macht dann beim nächsten Server weiter.

Konnte die Verbindung aufgebaut werden, referenziert `GetNISTTime` den `NetworkStream` der `TcpClient`-Instanz und überprüft (zur Sicherheit ...), ob dieser gelesen und geschrieben werden kann (was aber immer der Fall sein sollte). Da ein NIST-Server beim Daytime-Protokoll direkt nach dem Verbindungsaufbau Daten sendet, liest das Programm den Stream nach dem Verbindungsaufbau sofort aus und konvertiert die empfangenen Bytes in einen ASCII-String. Dieser String wird dann in einzelne Token (Bestandteile) zerlegt, wobei das Leerzeichen als Trennzeichen verwendet wird. Vor dem Auswerten der Token wird überprüft, ob die Anzahl in Ordnung ist: Beim Testen ist es vorgekommen, dass ein Server statt des erwarteten Zeit-Strings eine Folge von 0-Zeichen gesendet hat.

Das Token mit dem Index 5 ist dann der Health-Status. Dieser wird zunächst daraufhin überprüft, ob er den Wert »0« speichert (was ja auf die Rückgabe der genauen Zeit hinweist). Ist dies nicht der Fall, speichert `GetNISTTime` dies als Fehler in der Variablen `errors` und führt die Schleife weiter aus. Ist der Health-Status in Ordnung, werden das Datums- und das Zeit-Token in ihre Bestandteile zerlegt. Mit diesen Datums- bzw. Zeitteilen erzeugt `GetNISTTime` dann eine neue `DateTime`-Instanz und gibt diese in die lokale Zeit konvertiert zurück (womit die Schleife beendet ist). Tritt beim Lesen der Daten eine Ausnahme ein (z. B. weil der Timeout abgelaufen ist), wird die Fehlermeldung in der Variablen `errors` abgelegt. Für den Fall, dass alle Server unerreichbar sind, einen Health-Status ungleich »0« oder ungültige Daten zurückliefern oder nicht rechtzeitig Daten senden, erzeugt das Programm am Ende der Methode eine Ausnahme mit den ermittelten Fehlern als Meldung.

Die `TcpClient`-Instanz wird übrigens in der Schleife immer wieder neu erzeugt, weil die `Close`-Methode, die die Verbindung schließt, leider auch die internen Ressourcen freigibt, sodass ein zweites Öffnen einer Verbindung nicht mehr möglich ist.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Text` und `System.Net.Sockets` importieren.

```
public static System.DateTime GetNISTTime()
{
   // Variable für Fehlermeldungen
   string errors = null;

   // Array für die abzufragenden Server
   string[] servers = {"time-a.nist.gov", "time-b.nist.gov",
      "time.nist.gov", "utcnist.colorado.edu", "nist1.datum.com"};

   // Schleife, in der die Server abgefragt werden, bis das Ergebnis
   // in Ordnung ist
   for (int i = 0; i < servers.Length; i++)
   {
      TcpClient tcpClient = null;
      try
      {
         // TcpClient erzeugen und den Empfangs-Timeout auf eine Sekunde
         // setzen
         tcpClient = new TcpClient();
         tcpClient.ReceiveTimeout = 1000;

         // Versuch zum aktuellen Server eine Verbindung aufzubauen
         tcpClient.Connect(servers[i], 13);

         // Den NetworkStream referenzieren
         NetworkStream networkStream = tcpClient.GetStream();
```

Listing 104: Methode zur Ermittlung des offiziellen Datums von NIST-Servern

```
string result = null;
if (networkStream.CanWrite && networkStream.CanRead)
{
   // Das Ergebnis empfangen und in ASCII konvertieren
   byte[] bytes = new byte[tcpClient.ReceiveBufferSize];
   try
   {
      networkStream.Read(bytes, 0,
         (int) tcpClient.ReceiveBufferSize);
      result = Encoding.ASCII.GetString(bytes);
   }
   catch (Exception ex)
   {
      // Fehler dokumentieren
      if (errors != null) errors += "\r\n";
      errors += "Fehler bei der Abfrage von '" + servers[i] +
         ": " + ex.Message;
   }
}

if (result != null)
{
   // Das Ergebnis, das die Form JJJJJ YR-MO-DA HH:MM:SS TT L H
   // msADV UTC(NIST) OTM besitzt, in einzelne Token aufsplitten
   string[] token = result.Split(' ');

   // Anzahl der Token überprüfen
   if (token.Length >= 6)
   {
      // Den Health-Status auslesen und überprüfen
      string health = token[5];
      if (health == "0")
      {
         // Alles ok:  Datums- und Zeitangaben auslesen
         string[] dates = token[1].Split('-');
         string[] times = token[2].Split(':');

         // DateTime-Instanz mit diesen Daten erzeugen
         System.DateTime utcDate =
            new System.DateTime(Int32.Parse(dates[0]) + 2000,
            Int32.Parse(dates[1]), Int32.Parse(dates[2]),
            Int32.Parse(times[0]), Int32.Parse(times[1]),
            Int32.Parse(times[2]));
         // Lokale Zeit berechnen und zurückgeben
         return TimeZone.CurrentTimeZone.ToLocalTime(utcDate);
```

Listing 104: Methode zur Ermittlung des offiziellen Datums von NIST-Servern (Forts.)

```
                }
                else
                {
                   // Fehler dokumentieren
                   if (errors != null) errors += "\r\n";
                   errors += "Fehler bei der Abfrage von '" + servers[i] +
                      ": Der Health-Status ist " + health;
                }
             }
             else
             {
                // Fehler dokumentieren
                if (errors != null) errors += "\r\n";
                errors += "Fehler bei der Abfrage von '" + servers[i] +
                   ": Die Anzahl der Token ist kleiner als 6";
             }
          }
       }
       catch (Exception ex)
       {
          // Fehler dokumentieren
          if (errors != null) errors += "\r\n";
          errors += "Fehler bei der Abfrage von '" +
             servers[i] + ": " + ex.Message;
       }
       finally
       {
          try
          {
             // TcpClient schließen
             tcpClient.Close();
          }
          catch {}
       }
    }

    // Wenn die Methode hier ankommt, sind bei allen Abfragen
    // Fehler aufgetreten, also eine Ausnahme werfen
    throw new Exception(errors);
}
```

Listing 104: Methode zur Ermittlung des offiziellen Datums von NIST-Servern (Forts.)

Die Methode erfordert, dass eine Internetverbindung besteht und dass eine eventuelle Firewall die Anforderung des Programms zulässt.

Das Beispiel in Listing 105 liest das offizielle Datum ein und setzt damit das Systemdatum. Zum Setzen des Systemdatums wird die Methode `SetSystemDateTime` aus dem Rezept 69 eingesetzt.

```
Console.WriteLine("Hole das offizielle Datum von einem NIST-Server ...");
try
{
   // NIST-Server abfragen
   DateTime officialDate = GetNISTTime();

   // Systemdatum setzen
   SetSystemDateTime(officialDate);

   Console.WriteLine("Systemdatum auf den " +
      officialDate.ToString() + " gesetzt");
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Lesen oder " +
      "Setzen des Systemdatums: {0}", ex.Message);
}
```

Listing 105: Setzen des Systemdatums auf das von einem NIST-Server abgefragte offizielle Datum

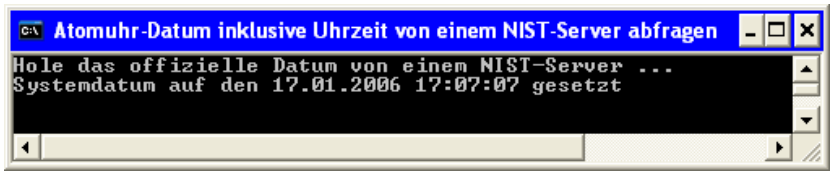


Abbildung 50: Das Programm hat das Systemdatum auf das offizielle Datum gesetzt (und ich habe endlich einmal die genaue Zeit ...)

71 Datum normalisieren

Viele Anwender wollen die Möglichkeit haben, ein Datum in der Form *ddmmyy*, *ddmmyyy* oder auch nur als *ddmm* einzugeben (z. B. *31122003* für den 31.12.2003 oder *3112* für den 31.12. des aktuellen Jahres). Die `Parse`-Methode der `DateTime`-Struktur erwartet ein Datum aber im aktuellen Systemformat oder im Format des am zweiten Argument übergebenen Objekts, das die `IFormatProvider`-Schnittstelle implementiert (normalerweise ist das ein `CultureInfo`-Objekt).

Die Methode `NormalizeDate` in Listing 106 ermöglicht die Eingabe von Datumswerten in typischen (Anwender-)Formaten und wandelt den übergebenen Datums-String in ein `DateTime`-Objekt um. Die folgenden Formate werden unterstützt:

- das normale Systemformat,
- *mm.dd*[.],
- *mm*[.],
- *ddmmyy*[*y*][*y*],

- *ddmm* und
- *d*[*d*].

Unvollständige Eingaben werden automatisch um das aktuelle Jahr bzw. das Jahr und den aktuellen Monat ergänzt. Aus *3112* wird im Jahr 2010 z. B. das Datum *31.12.2010*.

`NormalizeDate` entfernt zunächst eventuelle (störende) Whitespace-Zeichen am Anfang und am Ende des Datums-Strings und versucht dann, das Datum direkt zu konvertieren (falls es im Systemformat eingegeben wurde). Der `Parse`-Methode der `DateTime`-Struktur wird am zweiten Argument kein Format-Provider übergeben, damit die aktuell eingestellte Kultur verwendet wird.

Schlägt diese Konvertierung fehl, versucht `NormalizeDate` aus dem eingegebenen Datums-String einen gültigen Datums-String zu erzeugen. Bei der Abfrage der Datumsteile wird zwar das aktuelle System-Datumstrennzeichen berücksichtigt (damit eine Eingabe prinzipiell z. B. auch auf einem englischen Rechner ausgewertet werden kann), der neue Datums-String wird aber explizit im deutschen Format zusammengesetzt. Bei der Konvertierung am Ende der Methode übergibt das Programm dann auch am zweiten Argument der `Parse`-Methode ein neues `CultureInfo`-Objekt für die deutsche Kultur. So ist sichergestellt, dass der Datums-String auch auf Rechnern oder in Programmen konvertiert werden kann, die einer anderen Kultur zugeordnet sind.

`NormalizeDate` erreicht damit eine gewisse Globalisierung. Wenn das Datum allerdings nicht im Systemformat eingegeben wird, erwartet `NormalizeDate` die oben beschriebenen eher deutschen Eingabeformate (bei denen der Tag am Anfang und der Monat an zweiter Stelle steht). Ich habe versucht, eine Variante dieser Methode zu entwickeln, die die unterschiedlichen Kulturen berücksichtigt, sodass auf einem englischen Rechner das Datum z. B. auch im Format *mmddyyyy* eingegeben werden kann. Diese Variante wurde aber noch vor der Fertigstellung aufgrund der vielen verschiedenen Möglichkeiten viel zu komplex, sodass ich sie verworfen habe (damit noch Zeit fürs Snowboarden übrig blieb ☺).

`NormalizeDate` erfordert den Import des Namensraums `System`.

```
public static DateTime NormalizeDate(string dateString)
{
   // Whitespaces entfernen
   dateString = dateString.Trim();

   // Versuch das Datum direkt zu konvertieren
   try
   {
      return DateTime.Parse(dateString);
   }
   catch {}
```

Listing 106: Methode zur Normalisierung eines Datums

```
// Das aktuelle Datumstrennzeichen ermitteln
string dateSeparator =
   CultureInfo.CurrentCulture.DateTimeFormat.DateSeparator;

if (dateString.IndexOf(dateSeparator) == -1)
{
   // Wenn kein Punkt vorkommt: Versuch ein Datum im Format
   // ddmmyy[yy], ddmm oder d[d] zu erkennen
   if (dateString.Length >= 6 && dateString.Length <= 8)
   {
      // ddmmyy, ddmmyyy oder ddmmyyyy
      dateString = dateString.Substring(0, 2) + "." +
         dateString.Substring(2, 2) + "." +
         dateString.Substring(4, dateString.Length - 4);
   }
   else if (dateString.Length == 4)
   {
      // ddmm
      dateString = dateString.Substring(0, 2) + "." +
         dateString.Substring(2, 2) + "." + DateTime.Now.Year;
   }
   else if (dateString.Length <= 2)
   {
      // d oder dd
      if (dateString.Length == 1)
         dateString = "0" + dateString;
      dateString = dateString.Substring(0, 2) + "." +
         DateTime.Now.Month + "." + DateTime.Now.Year;
   }
}
else
{
   // Eingabe des Datums mit Punkten in der Form dd.mm oder
   // dd. Zunächst einen eventuellen rechten Punkt entfernen
   while (dateString.EndsWith(dateSeparator))
      dateString = dateString.Substring(0, dateString.Length - 1);

   // Ein im Format dd oder dd.mm angegebenes Datum um den
   // aktuellen Monat und das aktuelle Jahr ergänzen
   if (dateString.Length <= 2)
   {
      // dd.
      dateString = dateString + "." + DateTime.Now.Month + "." +
         DateTime.Now.Year;
   }
```

Listing 106: Methode zur Normalisierung eines Datums (Forts.)

```
      else
      {
         // dd.mm[.]
         dateString = dateString + "." + DateTime.Now.Year;
      }
   }

   // Versuch das ermittelte deutsche Datum zu konvertieren
   return DateTime.Parse(dateString,
      CultureInfo.CreateSpecificCulture("de"));
}
```

Listing 106: Methode zur Normalisierung eines Datums (Forts.)

Listing 107 zeigt ein kleines Testprogramm für die `NormalizeDate`-Methode, bei dem der Anwender in einer Schleife ein Datum mehrfach eingeben kann.

```
string dateString = null;
do
{
   // Den Anwender einen Datums-String eingeben lassen
   Console.Write("Geben Sie ein Datum ein (Beenden mit leerer Eingabe): ");
   dateString = Console.ReadLine();

   if (dateString != "")
   {

      try
      {
         // Das Datum normalisieren
         DateTime date = NormalizeDate(dateString);

         // Das normalisierte Datum ausgeben
         Console.WriteLine(date.ToString());
         Console.WriteLine();
      }
      catch (Exception ex)
      {
         Console.WriteLine(ex.Message);
      }
   }
} while (dateString != "");
```

Listing 107: Testprogramm für die NormalizeDate-Methode

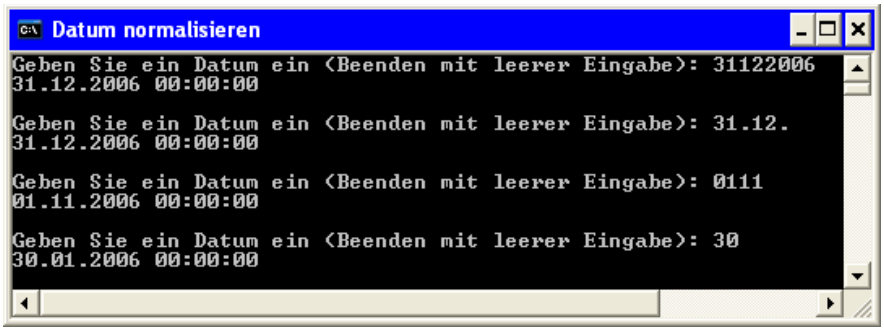


Abbildung 51: Das Testprogramm für die NormalizeDate-Methode

72 Deutsche Feiertage und andere besondere Tage berechnen

Das .NET Framework bietet keine Möglichkeiten, die deutschen Feiertage und die besonderen Tage (wie z. B. Rosenmontag) zu berechnen. Wenn Sie diese (z. B. für ein eigenes Kalender-Steuerelement) ermitteln wollen, müssen Sie selbst programmieren.

Einige Feiertage sind fest definiert:

- 1. Januar: Neujahr
- 6. Januar: Heilige drei Könige (nur Baden-Württemberg, Bayern und Sachsen-Anhalt)
- 1. Mai: Maifeiertag (Tag der Arbeit),
- 15. August: Maria Himmelfahrt (nur Saarland, Bayern und Gemeinden mit überwiegend katholischer Bevölkerung),
- 3. Oktober: Tag der deutschen Einheit,
- 31. Oktober: Reformationstag (nur Brandenburg, Mecklenburg-Vorpommern, Sachsen, Sachsen-Anhalt und Thüringen),
- 1. November: Allerheiligen (nur Baden-Württemberg, Bayern, Nordrhein-Westfalen, Rheinland-Pfalz und Saarland),
- 25. Dezember: Erster Weihnachtsfeiertag,
- 26. Dezember: Zweiter Weihnachtsfeiertag.

Hinzu kommen einige besondere Tage, die keine Feiertage sind:

- Heiliger Abend: 24. Dezember,
- Valentinstag: 14. Februar.

Einige der beweglichen Feiertage beziehen sich auf den Ostersonntag:

- Karfreitag: - 2 Tage
- Ostersonntag: +/- 0 Tage,

- Ostermontag: + 1 Tag,
- Christi Himmelfahrt: + 39 Tage,
- Pfingstsonntag: + 49 Tage,
- Pfingstmontag: + 50 Tage,
- Fronleichnam: + 60 Tage (Baden-Württemberg, Bayern, Hessen, Nordrhein-Westfalen, Rheinland-Pfalz, Saarland, Sachsen, Thüringen, außerdem Gemeinden mit überwiegend katholischer Bevölkerung).

Daneben existieren noch bewegliche, auf Ostern bezogene besondere Tage, die keine Feiertage sind:

- Aschermittwoch: - 46 Tage,
- Rosenmontag: - 48 Tage,
- Gründonnerstag: - 3 Tage.

Andere bewegliche Feiertage beziehen sich auf Weihnachten:

- 4. Advent: der Sonntag vor dem 25. Dezember,
- 1. bis 3. Advent: die jeweils vorigen Sonntage,
- Totensonntag: der Sonntag vor dem 1. Advent,
- Buß- & Bettag: der Mittwoch vor dem Totensonntag (nur Sachsen).

Das einzige Problem ist also die Berechnung des Datums für den Ostersonntag. Per Definition ist Ostern auf der nördlichen Halbkugel der Sonntag nach dem ersten Vollmond nach der Tag- und Nachtgleiche. Der Mathematiker Carl Friedrich Gauß hat dazu als Erster recht komplexe Formeln entwickelt. Leider arbeiten diese nicht korrekt, sodass für einige Jahre nachkorrigiert werden muss. Im Internet finden Sie eine Vielzahl weiterer Algorithmen zur Berechnung des Ostersonntags.

Ich habe den Algorithmus zur Oster-Berechnung natürlich nicht selbst entwickelt (ich bin kein Mathematiker ☺), sondern im Internet recherchiert. Der Algorithmus von Ronald W. Mallen soll nach Informationen aus dem Internet der definitive Algorithmus sein, der die Ostersonntage für den gregorianischen Kalender und die Jahre 1583 bis 4099 korrekt berechnet (vgl. *www.assa.org.au/bulletin/bennecma.pdf*). Sie finden diesen Algorithmus an der Adresse *www.assa.org.au/edm.html*. Ich habe ihn unverändert in der Methode `GetEasterSundayDate` nach C# übersetzt. Ronald W. Mallen hat die Veröffentlichung in dieser Übersetzung netterweise erlaubt. Für den Fall, dass ein ungültiges Jahr übergeben wird, generiert `GetEasterSundayDate` eine Ausnahme.

Zum Kompilieren der im Folgenden beschriebenen Methoden müssen Sie die Namensräume `System` und `System.Collections.Generic` importieren.

```
/* Methode zur Berechnung des Datums des Ostersonntags.
 * Nach dem Original von Ronald W. Mallen (www.assa.org.au/edm.html) */
public static DateTime GetEasterSundayDate(int year)
{
   // Überprüfen, ob das Jahr für die Osterberechnung gültig ist
   if (year < 1583 || year > 4099)
   {
      throw new Exception("Das Jahr muss zwischen 1583 " +
         "und 4099 liegen");
   }

   int firstDigits, remaining19, temp; // Zwischenergebnisse
   int tA, tB, tC, tD, tE; // Tabellenergebnisse A bis E

   firstDigits = year / 100; // die ersten zwei Ziffern
   remaining19 = year % 19; // Rest von year / 19

   // PFM-Datum berechnen
   temp = (firstDigits - 15) / 2 + 202 - 11 * remaining19;

   switch (firstDigits)
   {
      case 21:
      case 24:
      case 25:
      case 27:
      case 28:
      case 29:
      case 30:
      case 31:
      case 32:
      case 34:
      case 35:
      case 38:
         temp -= 1;
         break;
      case 33:
      case 36:
      case 37:
      case 39:
      case 40:
         temp -= 2;
         break;
   }

   temp = temp % 30;
```

Listing 108: Methode zur Berechnung des Datums des Ostersonntags

```
    tA = temp + 21;
    if (temp == 29)
    {
       tA = tA - 1;
    }
    if (temp == 28 && remaining19 > 10)
    {
       tA = tA - 1;
    }

    // Nächsten Sonntag ermitteln
    tB = (tA - 19) % 7;

    tC = (40 - firstDigits) % 4;
    if (tC == 3)
    {
       tC = tC + 1;
    }
    if (tC > 1)
    {
       tC = tC + 1;
    }

    temp = year % 100;
    tD = (temp + temp / 4) % 7;

    tE = ((20 - tB - tC - tD) % 7) + 1;

    // Das Datum ermitteln und zurückgeben
    int day = tA + tE;
    int month = 0;

    if (day > 31)
    {
       day -= 31;
       month = 4;
    }
    else
    {
       month = 3;
    }

    return new DateTime(year, month, day);
}
```

Listing 108: Methode zur Berechnung des Datums des Ostersonntags (Forts.)

Hinweis

Zur Absicherung meiner Übersetzung habe ich die auf der Seite *www.assa.org.au/edm.html* angegebenen Datumswerte für die Ostersonntage der Jahre 1700 bis 2299 in eine XML-Datei geschrieben und diese in einem Unit-Test mit den selbst berechneten Datumswerten für die Ostersonntage dieser Jahre verglichen.

Die deutschen speziellen Tage lassen sich nun auf der Basis des Osterdatums recht einfach berechnen. Die Methode `GetGermanSpecialDays` in Listing 112 macht genau das. Ich habe diese Methode so entwickelt, dass sie möglichst flexibel einsetzbar ist. `GetGermanSpecialDays` gibt deswegen ein spezielles Dictionary (Listing 111) zurück, das Instanzen der Klasse `GermanSpecialDay` (Listing 109) verwaltet. Über dieses Dictionary ist ein Durchgehen der besonderen Tage, aber auch ein gezieltes Zugreifen möglich. Zum Zugriff über den Indizierer habe ich eine Aufzählung erzeugt, die die deutschen Spezialtage als Konstante verwaltet (Listing 110). Das Ganze ist leider (bzw. zwangsläufig) ein wenig komplex ☺.

Zunächst folgt also die Klasse `GermanSpecialDay`, deren Instanzen einen besonderen Tag verwalten. Die Eigenschaften machen eine Aussage über den Namen des Tags, das Datum, darüber, ob der Tag deutschlandweit gilt und ob es sich um einen Feiertag handelt. Im Konstruktor werden zur späteren Vereinfachung alle Eigenschaftswerte übergeben. `GermanSpecialDay` implementiert zusätzlich die `CompareTo`-Methode der generischen `IComparable`-Schnittstelle. Diese Methode vergleicht die Datumsanteile der Objekte und ermöglicht ein Sortieren einer Liste dieser Objekte, z. B. über die `Sort`-Methode der `Array`-Klasse, wenn die Instanzen über ein Array verwaltet werden (was ich im Beispielprogramm in Listing 113 nutze):

```
public class GermanSpecialDay : IComparable<GermanSpecialDay>
{
   /* Der Schlüssel des speziellen Tags */
   public GermanSpecialDayKey Key;

   /* Der Name des speziellen Tags */
   public string Name;

   /* Das Datum des speziellen Tags */
   public DateTime Date;

   /* Gibt an, ob der spezielle Tag bundesweit gilt */
   public bool IsNationwide;

   /* Gibt an, ob es sich bei dem speziellen Tag */
   /* um einen Feiertag handelt */
   public bool IsHoliday;
   /* Konstruktor */
   public GermanSpecialDay(GermanSpecialDayKey key, string name,
      DateTime date, bool isNationwide, bool isHoliday)
   {
```

Listing 109: Klasse für einen speziellen Tag

```
      this.Key = key;
      this.Name = name;
      this.Date = date;
      this.IsNationwide = isNationwide;
      this.IsHoliday = isHoliday;
   }

   /* Vergleicht ein GermanSpecialDay-Objekt mit dem aktuellen */
   public int CompareTo(GermanSpecialDay otherSpecialDay)
   {
      return this.Date.CompareTo(otherSpecialDay.Date);
   }
}
```

Listing 109: Klasse für einen speziellen Tag (Forts.)

Die Aufzählung `GermanSpecialDayKey` beinhaltet Konstanten für die deutschen Spezialtage. Die Werte dieser Aufzählung werden später als Schlüssel der Auflistung verwendet:

```
public enum GermanSpecialDayKey
{
   Neujahr,
   HeiligeDreiKönige,
   Valentinstag,
   Maifeiertag,
   MariaHimmelfahrt,
   TagDerDeutschenEinheit,
   Reformationstag,
   Allerheiligen,
   HeiligerAbend,
   ErsterWeihnachtstag,
   ZweiterWeihnachtstag,
   Rosenmontag,
   Aschermittwoch,
   Gründonnerstag,
   Karfreitag,
   Ostersonntag,
   Ostermontag,
   ChristiHimmelfahrt,
   Pfingstsonntag,
   Pfingstmontag,
   Fronleichnam,
   ErsterAdvent,
   ZweiterAdvent,
   DritterAdvent,
```

Listing 110: Aufzählung für die besonderen Tage

```
   VierterAdvent,
   Totensonntag,
   BußUndBettag
}
```

Listing 110: Aufzählung für die besonderen Tage (Forts.)

Die Klasse `GermanSpecialDays` implementiert ein auf `GermanSpecialDay`-Objekte spezialisiertes Dictionary-Objekt und wird deswegen von der generischen `Dictionary`-Klasse (aus dem Namensraum `System.Collections.Generic`) abgeleitet. `GermanSpecialDays` implementiert neben einer Eigenschaft, die das Jahr verwaltet, eine spezialisierte `Add`-Methode, der alle Daten eines besonderen Tages übergeben werden. `Add` hängt ein mit diesen Daten neu erzeugtes `GermanSpecialDay`-Objekt an die von `Dictionary` geerbte Auflistung an.

```
public class GermanSpecialDays : Dictionary<GermanSpecialDayKey,
   GermanSpecialDay>
{
   private int year;
   /* Gibt das Jahr zurück, für das diese speziellen Tage gelten */
   public int Year
   {
      get { return this.year; }
   }

   /* Konstruktor */
   internal GermanSpecialDays(int year)
   {
      this.year = year;
   }

   /* Fügt der Auflistung ein neues GermanSpecialDay-Objekt hinzu */
   internal void Add(GermanSpecialDayKey key, string name, DateTime
      date, bool nationWide, bool holiday)
   {
      base.Add(key, new GermanSpecialDay(key, name, date,
         nationWide, holiday));
   }
}
```

Listing 111: Dictionary für die besonderen Tage

Die Methode `GetGermanSpecialDays` ermittelt schließlich die deutschen Spezialtage in einer `GermanSpecialDays`-Instanz. Die Berechnung erfolgt nach dem anfangs dieses Rezepts beschriebenen Schema, weswegen ich auf eine weitere Beschreibung verzichte. Die Kommentare im Quellcode sagen alles Wichtige aus:

```
public static GermanSpecialDays GetGermanSpecialDays(int year)
{
   // GermanSpecialDays-Instanz erzeugen
   GermanSpecialDays gsd = new GermanSpecialDays(year);
   // Die festen besonderen Tage eintragen
   gsd.Add(GermanSpecialDayKey.Neujahr, "Neujahr", new DateTime(year, 1, 1),
      true, true);
   gsd.Add(GermanSpecialDayKey.HeiligeDreiKönige,
      "Heilige Drei Könige", new DateTime(year, 1, 6), false, true);
   gsd.Add(GermanSpecialDayKey.Valentinstag, "Valentinstag", new
      DateTime(year, 2, 14), true, false);
   gsd.Add(GermanSpecialDayKey.Maifeiertag, "Maifeiertag", new DateTime
      (year, 5, 1), true, true);
   gsd.Add(GermanSpecialDayKey.MariaHimmelfahrt, "Maria Himmelfahrt",
      new DateTime(year, 8, 15), false, true);
   gsd.Add(GermanSpecialDayKey.TagDerDeutschenEinheit,
      "Tag der Deutschen Einheit", new DateTime(year, 10, 3), true, true);
   gsd.Add(GermanSpecialDayKey.Reformationstag, "Reformationstag", new
      DateTime(year, 10, 31), false, true);
   gsd.Add(GermanSpecialDayKey.Allerheiligen, "Allerheiligen", new
      DateTime(year, 11, 1), false, true);
   gsd.Add(GermanSpecialDayKey.HeiligerAbend, "Heiliger Abend", new
      DateTime(year, 12, 24), true, true);
   gsd.Add(GermanSpecialDayKey.ErsterWeihnachtstag,
      "Erster Weihnachtstag", new DateTime(year, 12, 25), true, true);
   gsd.Add(GermanSpecialDayKey.ZweiterWeihnachtstag,
      "Zweiter Weihnachtstag", new DateTime(year, 12, 26), true, true);

   // Datum des Ostersonntag berechnen
   DateTime easterSunday = GetEasterSundayDate(year);

   // Die beweglichen besonderen Tage ermitteln, die sich auf Ostern beziehen
   gsd.Add(GermanSpecialDayKey.Rosenmontag, "Rosenmontag",
      easterSunday.AddDays(-48), true, false);
   gsd.Add(GermanSpecialDayKey.Aschermittwoch, "Aschermittwoch",
      easterSunday.AddDays(-46), true, false);
   gsd.Add(GermanSpecialDayKey.Gründonnerstag, "Gründonnerstag",
      easterSunday.AddDays(-3), true, false);
   gsd.Add(GermanSpecialDayKey.Karfreitag, "Karfreitag",
      easterSunday.AddDays(-2), true, true);
   gsd.Add(GermanSpecialDayKey.Ostermontag, "Ostermontag",
      easterSunday.AddDays(1), true, true);
   gsd.Add(GermanSpecialDayKey.Ostersonntag, "Ostersonntag",
      easterSunday, true, true);
   gsd.Add(GermanSpecialDayKey.ChristiHimmelfahrt,
      "Christi Himmelfahrt", easterSunday.AddDays(39), true, true);
```

Listing 112: Methode zur Berechnung der besonderen Tage in Deutschland

```
gsd.Add(GermanSpecialDayKey.Pfingstsonntag, "Pfingstsonntag",
   easterSunday.AddDays(49), true, true);
gsd.Add(GermanSpecialDayKey.Pfingstmontag, "Pfingstmontag",
   easterSunday.AddDays(50), true, true);
gsd.Add(GermanSpecialDayKey.Fronleichnam, "Fronleichnam",
   easterSunday.AddDays(60), false, true);

// Die beweglichen besonderen Tage ermitteln, die sich auf Weihnachten
// beziehen
// Sonntag vor dem 25. Dezember (4. Advent) ermitteln
DateTime firstXMasDay = new DateTime(year, 12, 25);
DateTime fourthAdvent;
int weekday = (int)firstXMasDay.DayOfWeek;
if (weekday == 0)
{
   // Sonntag
   fourthAdvent = firstXMasDay.AddDays(-7);
}
else
{
   fourthAdvent = firstXMasDay.AddDays(-weekday);
}
gsd.Add(GermanSpecialDayKey.VierterAdvent, "Vierter Advent",
   fourthAdvent, true, false);
gsd.Add(GermanSpecialDayKey.DritterAdvent, "Dritter Advent",
   fourthAdvent.AddDays(-7), true, false);
gsd.Add(GermanSpecialDayKey.ZweiterAdvent, "Zweiter Advent",
   fourthAdvent.AddDays(-14), true, false);
gsd.Add(GermanSpecialDayKey.ErsterAdvent, "Erster Advent",
   fourthAdvent.AddDays(-21), true, false);

// Totensonntag ermitteln
DateTime deadSunday = fourthAdvent.AddDays(-28);
gsd.Add(GermanSpecialDayKey.Totensonntag, "Totensonntag", deadSunday,
   true, false);

// Den Mittwoch vor dem Totensonntag ermitteln
weekday = (int)deadSunday.DayOfWeek;
if (weekday < 4)
{
   // Sonntag bis Mittwoch
   gsd.Add(GermanSpecialDayKey.BußUndBettag, "Buß- und Bettag",
      deadSunday.AddDays(-(weekday + 4)), false, true);
}
else
{
```

Listing 112: Methode zur Berechnung der besonderen Tage in Deutschland (Forts.)

```
      // Donnerstag bis Samstag
      gsd.Add(GermanSpecialDayKey.BußUndBettag, "Buß- und Bettag",
         deadSunday.AddDays(-(weekday - 3)), false, true);
   }

   // Das Ergebnis zurückgeben
   return gsd;
}
```

Listing 112: Methode zur Berechnung der besonderen Tage in Deutschland (Forts.)

Das Programm erfordert den Import des Namensraums `System.Collections`.

Listing 113 zeigt eine beispielhafte Anwendung der `GetGermanSpecialDays`-Methode. Berechnet werden die besonderen Tage des aktuellen Jahres. Das Programm liest zuerst gezielt die Daten des Ostersonntags aus. Danach gibt das Programm die ermittelten Daten der Tage für die vier Bereiche aus (bundesweite Feiertage, lokale Feiertage etc.). Da die Ausgabe sortiert erfolgen soll (bei `Hashtable`-ähnlichen Auflistungen sind die Daten immer unsortiert und entsprechen auch nicht der Reihenfolge des Hinzufügens), erzeugt das Programm ein einfaches Array mit der Anzahl der ermittelten Feiertage, kopiert über die `CopyTo`-Methode der `Values`-Auflistung die Werte der ermittelten `GermanSpecialDays`-Instanz in dieses Array und sortiert das Array über `Array.Sort`, bevor dieses in einer Schleife durchlaufen wird. Das Sortieren erfolgt nach dem Datum, da die im Array verwalteten `GermanSpecialDay`-Objekte die `CompareTo`-Methode der `IComparable`-Schnittstelle implementieren und diese die Datumswerte vergleicht.

```
// Feiertage des aktuellen Jahrs berechnen
GermanSpecialDays gsd = GetGermanSpecialDays(DateTime.Now.Year);

// Ostersonntag ermitteln
GermanSpecialDay easterSunday = gsd[GermanSpecialDayKey.Ostersonntag];
Console.WriteLine(easterSunday.Name);
Console.WriteLine(easterSunday.Date.ToShortDateString());
Console.WriteLine();

// Die speziellen Tage zur Sortierung in ein Array
// schreiben und sortieren
GermanSpecialDay[] outputArray = new GermanSpecialDay[gsd.Count];
gsd.Values.CopyTo(outputArray, 0);
Array.Sort(outputArray, 0, outputArray.Length);

// Die bundesweiten Feiertage ausgeben
Console.WriteLine("Bundesweite Feiertage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
   if (outputArray[i].IsHoliday && outputArray[i].IsNationwide)
   {
```

Listing 113: Beispielanwendung der GetGermanSpecialDays-Methode

```
      Console.WriteLine(outputArray[i].Name);
      Console.WriteLine(outputArray[i].Date.ToShortDateString());
      Console.WriteLine();
   }
}

// Die lokalen Feiertage ausgeben
Console.WriteLine("Lokale Feiertage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
   if (outputArray[i].IsHoliday && outputArray[i].IsNationwide == false)
   {
      Console.WriteLine(outputArray[i].Name);
      Console.WriteLine(outputArray[i].Date.ToShortDateString());
      Console.WriteLine();
   }
}

// Die bundesweiten Spezialtage ausgeben
Console.WriteLine("Bundesweite Spezialtage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
   if (outputArray[i].IsHoliday == false && outputArray[i].IsNationwide)
   {
      Console.WriteLine(outputArray[i].Name);
      Console.WriteLine(outputArray[i].Date.ToShortDateString());
      Console.WriteLine();
   }
}

// Die lokalen Spezialtage ausgeben
Console.WriteLine("Lokale Spezialtage\r\n");
for (int i = 0; i < outputArray.Length; i++)
{
   if (outputArray[i].IsHoliday == false &&
      outputArray[i].IsNationwide == false)
   {
      Console.WriteLine(outputArray[i].Name);
      Console.WriteLine(outputArray[i].Date.ToShortDateString());
      Console.WriteLine();
   }
}
```

Listing 113: Beispielanwendung der GetGermanSpecialDays-Methode (Forts.)

Zum weiteren Testen der `GetGermanSpecialDays`-Methode habe ich die besonderen Tage der Jahre 2002 bis 2010 berechnet und mit den offiziellen Datumswerten verglichen. Was mich erstaunte, war die Tatsache, dass bereits die erste Version fehlerfrei war ☺.

Abbildung 52 zeigt die Ausgabe dieses Programms für das Jahr 2006.

Hinweis

In Rezept 254 zeige ich, wie Sie die berechneten Feiertage im `MonthCalendar`-Steuerelement darstellen.

73 Ermitteln, ob ein bestimmter Tag ein Feiertag ist

Basierend auf der Methode `GetGermanSpecialDays` aus dem Rezept 72 können Sie recht einfach ermitteln, ob ein bestimmter Tag ein Feiertag ist. Die Methode `IsGermanHoliday` in Listing 114 erzeugt dazu eine Auflistung der besonderen Tage des Jahres des abgefragten Datums und geht diese durch. Handelt es sich um einen Feiertag, vergleicht die Methode den Tag und den Monat der `DateTime`-Instanzen (das Jahr ist ja auf jeden Fall identisch). Wird ein Feiertag ermittelt, gibt die Methode `true` zurück und schreibt den Namen des Feiertags und eine Information darüber, ob es ein bundesweiter Feiertag ist, in die am zweiten und dritten Argument als `out`-Referenz übergebenen Variablen.



Abbildung 52: Berechnung der Feiertage und der besonderen Tage für das Jahr 2006

IsGermanHoliday erfordert den Import der Namensräume System und System.Collections.Generic.

```
/* Die Implementierung der Klassen, Aufzählung und der Methoden für die
 * Oster- und Feiertagsberechnung wird aus Übersichtsgründen hier nicht
 * dargestellt */

public static bool IsGermanHoliday(this DateTime date, out string name,
   out bool isNationWide)
{
   // out-Argumente initialisieren
   name = null;
   isNationWide = false;
   // Auflistung der besonderen Tage des angegebenen Jahres erzeugen,
   // durchgehen und das Datum der Feiertage mit dem angegebenen Datum
   // vergleichen
   foreach (GermanSpecialDay gsd in GetGermanSpecialDays(date.Year).Values)
   {
      if (date.Day == gsd.Date.Day &&
         date.Month == gsd.Date.Month)
      {
         // Datum gefunden
         if (gsd.IsHoliday)
         {
            // Es ist ein Feiertag: Infos definieren und true zurückgeben
            name = gsd.Name;
            isNationWide = gsd.IsNationwide;
            return true;
         }
         else
         {
            // Kein Feiertag
            return false;
         }
      }
   }

   // Tag wurde nicht gefunden
   return false;
}
```

Listing 114: Methode zur Ermittlung, ob ein bestimmter Tag ein Feiertag ist

Listing 115 zeigt eine beispielhafte Anwendung der IsGermanHoliday-Methode, die überprüft, ob der 1.11. des aktuellen Jahres ein Feiertag ist.

```
DateTime date = new DateTime(DateTime.Now.Year, 11, 1);

string name;
bool nationWide;
if (date.IsGermanHoliday(out name, out nationWide))
{
   if (nationWide)
   {
      Console.WriteLine("Der {0} ist ein bundesweiter Feiertag: {1}",
         date.ToShortDateString(), name);
   }
   else
   {
      Console.WriteLine("Der {0} ist ein lokaler Feiertag: {1}",
         date.ToShortDateString(), name);
   }
}
else
{
   Console.WriteLine("Der {0} ist kein Feiertag",
      date.ToShortDateString());
}
```

Listing 115: Ermitteln, ob der 1.11. des aktuellen Jahres ein Feiertag ist

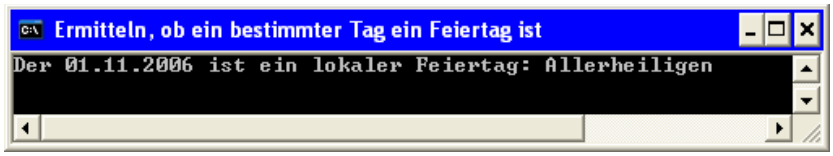


Abbildung 53: Das Ergebnis der Ermittlung, ob der 1.11. ein Feiertag ist

74 Zeit genau messen

Das genaue Messen eines Zeitraums ist besonders dann interessant, wenn Sie Ihre Methoden oder Algorithmen daraufhin überprüfen wollen, wie schnell diese ausgeführt werden, oder wenn Sie die Performance verschiedener Algorithmen miteinander vergleichen wollen. Dabei kommt es häufig darauf an, dass die Zeit mit einer möglichst hohen Genauigkeit und Auflösung gemessen wird.

Zur groben Zeitmessung können Sie die Ticks der aktuellen Zeit verwenden: Das Systemdatum wird in Einheiten von 100ns-Intervallen gespeichert, die seit dem 1.1.0001 00:00:00 vergangen sind. Nun könnten Sie denken, dass ein 100ns-Intervall, das ja immerhin 0,1 Mikrosekunden bzw. 0,0000001 Sekunden entspricht, für eine (recht) genaue Zeitmessung ausreicht. Und das würde es auch, wäre da nicht die Tatsache, dass der System-Zeitgeber je nach System lediglich alle 15 bis 55 Millisekunden aktualisiert wird. Auf einem High-Performance-System mit Windows ab Version 2000 können Sie die Zeit also mit einer Genauigkeit von 15 Millise-

kunden messen, auf einem älteren System leider nur mit einer höheren Ungenauigkeit. Als Beweis können Sie den folgenden Code verwenden:

```
// Die aktuellen Ticks zwischenspeichern
long startTicks = DateTime.Now.Ticks;

// Schleife, die nur wenig Zeit benötigt
int i = 0;
while (i < 10000000)
{
   i++;
}

// Die benötigten Ticks und Sekunden berechnen und ausgeben
long ticks = (DateTime.Now.Ticks - startTicks);
double seconds = ticks / 10000000F;

Console.WriteLine("Benötigte Zeit: {0} Ticks, {1} Sekunden", ticks, seconds);
```

Listing 116: Ungenaue Zeitmessung über die Ticks der Systemzeit

Wenn Sie die Schleifenanzahl auf Ihrem System etwas variieren, erhalten Sie bei einer niedrigeren Anzahl an Durchläufen als Ergebnis 0 und bei einer etwas höheren als Ergebnis einen Wert, der der Aktualisierung des Zeitgebers Ihres Systems entspricht (auf meinem System war das der Wert 0,015625 Sekunden). Bei einem Rechner mit weniger als 2 GHz Prozessorfrequenz müssen Sie die Anzahl der Durchläufe u. U. verkleinern um an dieses Ergebnis zu kommen. Da die gemessenen Werte immer entweder 0 oder ein Vielfaches des Aktualisierungsintervalls sind, ist bewiesen, dass der Systemtimer tatsächlich ein so hohes Intervall verwendet.

Die Lösung des Zeitmess-Problems liegt ab der Version 2.0 des .NET Framework in der Verwendung der `Stopwatch`-Klasse aus dem Namensraum `System.Diagnostics`. Eine Instanz dieser Klasse verwendet zur Zeitmessung den Performance-Counter[3] des Mainboards, sofern dieses einen solchen besitzt. Besitzt das Mainboard keinen Performance-Counter, verwendet die `Stopwatch`-Klasse den (ungenauen) System-Timer. Ob die hohe Auflösung zur Verfügung steht, können Sie über die statische Eigenschaft `IsHighResolution` abfragen.

Die Zeitmessung an sich ist einfach: Starten Sie diese über die `Start`-Methode und beenden Sie die Messung über `Stop`. Über die Eigenschaften `Elapsed`, `ElapsedMilliseconds` und `ElapsedTicks` können Sie die benötigte Zeit abfragen.

Bei der Verwendung von `ElapsedTicks` müssen Sie aufpassen: Mit »Ticks« sind hier auf einem Mainboard mit Performance-Counter nicht die in .NET üblichen 100ns-Intervalle gemeint, sondern Intervalle, die sich aus 1s geteilt durch die aktuelle Frequenz (die Sie aus `Stopwatch.Frequency` auslesen können) ergeben. Deshalb ist diese Eigenschaft ideal zur hochgenauen Messung, da Sie die aktuelle Frequenz mit in die Berechnung des Ergebnisses einbeziehen:

3. Ein Counter ist für Programmierer ein Wert, der in regelmäßigen oder unregelmäßigen Abständen hochgezählt wird.

```
// Stopwatch erzeugen und starten
Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();

// Zu messende Aktion ausführen

// Stopwatch stoppen und das Ergebnis berechnen
stopwatch.Stop();
double seconds = stopwatch.ElapsedTicks / (float)Stopwatch.Frequency;

// Reset ausführen, damit beim nächsten Start
// wieder ab null gezählt wird
stopwatch.Reset();
```

Bei der weiteren Verwendung einer `Stopwatch`-Instanz müssen Sie beachten, dass die `Start`-Methode die bisher gemessene Zeit nicht zurücksetzt. Wenn Sie diese Methode nach einer Messung erneut aufrufen, wird die gemessene Zeit der bisher gemessenen aufaddiert. Um die Stoppuhr zurückzusetzen, können Sie aber einfach die `Reset`-Methode aufrufen.

Listing 117 implementiert ein kleines Programm zum Testen beider Varianten der Zeitmessung. Gemessen wird die Zeit, die eine `while`-Schleife für 1.000.000 Durchläufe benötigt, wenn in dieser eine `int`-Variable hochgezählt wird.

```
// Zeit ungenau (mit einem Intervall von etwa 15 bis 55ms) über die Ticks
// der DateTime-Struktur messen
Console.WriteLine("Zeitmessung über DateTime.Now.Ticks:");
long startTicks = DateTime.Now.Ticks;

int i = 0;
while (i < 1000000)
{
   i++;
}

long ticks = (DateTime.Now.Ticks - startTicks);
double seconds = ticks / 10000000F;
Console.WriteLine("Benötigte Zeit: {0} Ticks, {1} Sekunden", ticks, seconds);

// Zeit genau über eine Stopwatch-Instanz messen
Console.WriteLine();
Console.WriteLine("Zeitmessung über Stopwatch:");
Stopwatch stopwatch = new Stopwatch();
Console.WriteLine("Hohe Auflösung: {0}", Stopwatch.IsHighResolution);
Console.WriteLine("Aktuelle Frequenz: {0}", Stopwatch.Frequency);
stopwatch.Start();

i = 0;
while (i < 1000000)
```

Listing 117: Ungenaue und genaue Zeitmessung einer Schleife mit 1.000.000 Durchläufen

```
{
   i++;
}
stopwatch.Stop();
seconds = stopwatch.ElapsedTicks / (float)Stopwatch.Frequency;
Console.WriteLine("Benötigte Zeit: {0} Sekunden", seconds);
stopwatch.Reset();
```

Listing 117: Ungenaue und genaue Zeitmessung einer Schleife mit 1.000.000 Durchläufen (Forts.)

Abbildung 54 zeigt, dass die Zeitmessung über eine `Stopwatch`-Instanz wesentlich genauer ist als die über die Systemticks.



Abbildung 54: Zeitmessung einer Schleife mit 1.000.000 Durchläufen

75 Eine Eingabe daraufhin überprüfen, ob diese ein Datum ergeben kann

In einem meiner Projekte habe ich eine spezielle `TextBox` entwickelt, die u. a. die Eingabe daraufhin beschränken sollte, dass nur die Eingabe eines potenziellen Datums (mit oder ohne Zeitangabe) möglich ist. Das Problem dabei war, dass bei der Eingabe der ersten Zeichen das Datum natürlich noch nicht komplett war und somit auch nicht über `DateTime.TryParse` geprüft werden konnte.

Also habe ich für die Überprüfung der Eingabe eine Methode entwickelt, die mit regulären Ausdrücken arbeitet. Die einzelnen regulären Ausdrücke überprüfen die Eingabe in einzelnen Alternativen. Für ein deutsches Datum beginnt die Überprüfung z. B. mit dem (gruppierten) Ausdruck »(?<day>\d{1,2})«, also dem Test auf ein oder zwei Dezimalziffern in einer benannten Gruppe. Die nächste Alternative wäre dann »(?<day>\d{1,2})\.)«, also der Test auf zwei Dezimalziffern, gefolgt von einem Punkt. Die letzte Alternative ist »((?<day>\d{1,2})\.(?<month>\d{1,2})\.(?<year>\d{2,4})(?<hour>\d{1,2}):(?<minute>\d{1,2}): (?<second>\d{1,2}))« für den Fall, dass die Eingabe der Zeit bei der Überprüfung erlaubt ist.

Damit ist die Überprüfung natürlich noch nicht beendet, denn die einzelnen Ziffernblöcke müssen auch noch auf den erwarteten Bereich (1-31 Tage, 1-12 Monate, 0-24 Stunden, 0-60 Minuten etc.) überprüft werden. Dazu habe ich die einzelnen Gruppen des Ergebnisses des regulären Ausdrucks ausgewertet. Da die Gruppen benannt sind, kann das Programm einfach über den Namen der Gruppe auf den jeweiligen Wert zugreifen, unabhängig davon, welche der Alternativen zu dem zu prüfenden String passt.

Listing 118 zeigt das Ergebnis dieser Überlegungen.

Das in der Praxis größte Problem war allerdings die internationale Unterstützung. In verschiedenen Ländern wird ein Datum auf sehr unterschiedliche Weise eingegeben, wobei die Formate allerdings in (grobe) Gruppen eingeteilt werden können. Die Gruppe, die das bei uns verwendete Datumsformat einsetzt, besteht z. B. aus den Ländern Deutschland, Aserbaidschan, Finnland, der Schweiz und vielen anderen. Das englische Format wird natürlich in Großbritannien, aber auch u. a. in vielen arabischen Ländern und spanischsprachigen Ländern verwendet.

Leider liefert das .NET Framework keine Unterstützung für diese Gruppierung. Ich musste die einzelnen Gruppen selbst herausfinden, wozu ich einfach je ein Datum für alle Kulturen ausgegeben und die einzelnen Gruppen dann »von Hand« zusammengefasst habe.

Herausgekommen ist ein ziemlich umfangreicher Quellcode, den ich nicht in das Buch übernommen habe, weil er in etwa 10 Seiten benötigen würde. Deshalb sehen Sie in Listing 118 nur die einfache deutsche Variante.

Hinweis

Den kompletten Quellcode mit internationaler Unterstützung finden Sie im Repository, in der Codebook-Klassenbibliothek und im Beispiel zu diesem Rezept. Die Methode `IsPotentialDateTime`, die Sie dort finden, hat übrigens aufgrund der vielen unterschiedlichen Datumsformate mehrere Tage Entwicklung gekostet. Ich hoffe, dass sie auch verwendet wird ☺. Die Methode scheint auf jeden Fall fehlerfrei zu sein: Der in die Codebook-Klassenbibliothek-Solution eingebettete Unit-Test (der zugegebenermaßen nur positive Fälle prüft) meldet keine Probleme.

Die Argumente der `IsPotentialGermanDate`-Methode haben die folgende Bedeutung:

- input: Die zu prüfende Eingabe
- includeTime: Bestimmt, ob auch die Eingabe einer Zeit erlaubt ist
- allowZeroForDayMonth: Gibt an, ob für den Tag und den Monat auch eine führende Null eingegeben werden kann (Viele Anwender sind es gewohnt, Tage und Monate kleiner als 10 mit einer führenden Null einzugeben).

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Text.RegularExpressions` einbinden.

```
public static bool IsPotentialGermanDate(this string input,
   bool includeTime, bool allowZeroForDayMonth)
{
   // Die aktuellen Separatoren setzen
   string dateSeparator = @"\.";
   string timeSeparator = ":";
```

Listing 118: Methode zur Überprüfung einer Eingabe daraufhin, ob diese ein gültiges deutsches Datum ergeben kann

```
// Minimalen Tag und Monat ermitteln
int dayMin = (allowZeroForDayMonth ? 0 : 1);
int monthMin = (allowZeroForDayMonth ? 0 : 1);

// Das Muster für den regfulären Ausdruck zusammensetzen
string pattern =
   @"(?<day>\d{1,2})|" +
   @"(?:(?<day>\d{1,2})" + dateSeparator + @")|" +
   @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2}))|" +
   @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
      dateSeparator + @")|" +
   @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
      dateSeparator + @"(?<year>\d{1,4}))";

if (includeTime)
{
   // Datum mit Zeit
   pattern += "|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) )|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) (?<hour>\d{1,2}))|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) (?<hour>\d{1,2})" +
         timeSeparator + @")|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) (?<hour>\d{1,2})" +
         timeSeparator + @"(?<minute>\d{1,2}))|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) (?<hour>\d{1,2})" +
         timeSeparator + @"(?<minute>\d{1,2})" + timeSeparator + @")|" +
      @"(?:(?<day>\d{1,2})" + dateSeparator + @"(?<month>\d{1,2})" +
         dateSeparator + @"(?<year>\d{1,4}) (?<hour>\d{1,2})" +
         timeSeparator + @"(?<minute>\d{1,2})" +
         timeSeparator + @"(?<second>\d{1,2}))";
}
pattern = "^(?:" + pattern + ")$";

// Den regulären Ausdruck prüfen
Match match = Regex.Match(input, pattern);

if (match.Success)

{
```

Listing 118: Methode zur Überprüfung einer Eingabe daraufhin, ob diese ein gültiges deutsches Datum ergeben kann (Forts.)

```
      // Der reguläre Ausdruck führte grundlegend zum Erfolg:
      // Die einzelnen Werte auf die Einhaltung der Grenzen
      // testen
      bool valid = false;
      if (match.Groups["day"].Value.Length > 0) // Tag
      {
         int day = Convert.ToInt32(match.Groups["day"].ToString());
         valid = (day >= dayMin && day <= 31);
         if (match.Groups["month"].Value.Length > 0) // Monat
         {
            int month = Convert.ToInt32(match.Groups["month"].ToString());
            valid &= (month >= monthMin && month <= 12);
            if (match.Groups["year"].Value.Length > 0) // Jahr
            {
               int year = Convert.ToInt32(match.Groups["year"].ToString());
               valid &= (year >= 0 && year <= 9999);
               if (match.Groups["hour"].Value.Length > 0) // Stunde
               {
                  int hour =
                     Convert.ToInt32(match.Groups["hour"].ToString());
                  valid &= (hour >= 0 && hour <= 23);
                  if (match.Groups["minute"].Value.Length > 0) // Minute
                  {
                     int minute =
                        Convert.ToInt32(match.Groups["minute"].ToString());
                     valid &= (minute >= 0 && minute <= 59);
                     if (match.Groups["second"].Value.Length > 0) // Sekunde
                     {
                        int second =
                           Convert.ToInt32(match.Groups["second"].ToString());
                        valid &= (second >= 0 && second <= 59);
                     }
                  }
               }
            }
         }
      }
      return valid;
   }
   else
   {
    return false;
   }
}
```

Listing 118: Methode zur Überprüfung einer Eingabe daraufhin, ob diese ein gültiges deutsches Datum ergeben kann (Forts.)

76 Eine Eingabe daraufhin überprüfen, ob diese eine Zeit ergeben kann

Die Überprüfung einer Eingabe daraufhin, ob diese eine gültige Zeit ergeben kann, entspricht im Wesentlichen der Datums-Überprüfung aus dem Rezept 75. Der Unterschied ist lediglich, dass hier kein Datumsteil vorhanden ist. Das Rezept wird also um die Datumsprüfung reduziert.

Wie auch in Rezept 75 war die Globalisierung das Problem bei der Zeiteingabeprüfung. Zeitangaben werden zwar in allen Ländern in demselben Basis-Format angegeben, einige Länder geben die Stunden aber nicht in unserem 24-Stunden-, sondern im 12-Stunden-Format an. Ein dem amerikanischen Begriff »AM« (lat.: Ante meridiem) entsprechender Begriff steht für den Vormittag, ein dem »PM« (lat.: Post meridiem) entsprechender für den Nachmittag. Problematisch ist zudem, dass bei einigen Kulturen der AM/PM-Wert vor der Zeitangabe steht.

Um Eingaben im 12-Stunden-Format zu unterstützen habe ich die in Rezept 75 entwickelten Datumsformat-Gruppen verwendet. Dieses und die notwendige Unterscheidung der Überprüfung haben den Quellcode wieder zu lang werden lassen, um diesen im Buch darzustellen. In Listing 119 sehen Sie deswegen nur die vereinfachte Überprüfung auf eine Zeit im 24-Stunden-Format.

Hinweis

Im Repository, in der Codebook-Klassenbibliothek und im Beispiel zu diesem Rezept finden Sie natürlich die vollständige Version in Form der `IsPotentialDateTime`-Methode, die auch in der Lage ist, eine Zeitangabe ohne Datum zu prüfen.

Die Methode `IsPotential24HourTime` erfordert das Importieren der Namensräume `System` und `System.Text.RegularExpressions`.

```
public static bool IsPotential24HourTime(this string input)
{
   // Die aktuellen Separatoren setzen
   string timeSeparator = ":";

   string pattern = "^(?:" +
      @"(?:(?<hour>\d{1,2}))|" +
      @"(?:(?<hour>\d{1,2})" + timeSeparator + @")|" +
      @"(?:(?<hour>\d{1,2})" + timeSeparator + @"(?<minute>\d{1,2}))|" +
      @"(?:(?<hour>\d{1,2})" + timeSeparator + @"(?<minute>\d{1,2})" +
        timeSeparator + @")|" +
      @"(?:(?<hour>\d{1,2})" + timeSeparator + @"(?<minute>\d{1,2})" +
         timeSeparator + @"(?<second>\d{1,2}))" +
      ")$";
```

Listing 119: Methode zur Überprüfung einer Eingabe daraufhin, ob diese potenziell eine gültige Zeit im 24-Stunden-Format ergeben kann

```
   // Den regulären Ausdruck prüfen
   Match match = Regex.Match(input, pattern);

   if (match.Success)
   {
      // Der reguläre Ausdruck führte grundlegend zum Erfolg:
      // Die einzelnen Werte auf die Einhaltung der Grenzen
      // testen
      int hour = Convert.ToInt32(match.Groups["hour"].ToString());
      bool valid = (hour >= 0 && hour <= 23);
      if (match.Groups["minute"].Value.Length > 0) // Minute
      {
         int minute = Convert.ToInt32(match.Groups["minute"].ToString());
         valid &= (minute >= 0 && minute <= 59);
         if (match.Groups["second"].Value.Length > 0) // Sekunde
         {
            int second =
               Convert.ToInt32(match.Groups["second"].ToString());
            valid &= (second >= 0 && second <= 59);
         }
      }
      return valid;
   }
   else
   {
      return false;
   }
}
```

Listing 119: Methode zur Überprüfung einer Eingabe daraufhin, ob diese potenziell eine gültige Zeit im 24-Stunden-Format ergeben kann (Forts.)

77 Datumswerte fehlerfrei zwischen Systemen mit verschiedenen Zeitzonen austauschen

Wenn Sie Datumswerte zwischen Systemen austauschen wollen, die verschiedene Zeitzonen verwenden, können Sie dafür nicht unbedingt eine `DateTime`-Instanz einsetzen.

Ein `DateTime`-Objekt besitzt einen Typ, den Sie über die `Kind`-Eigenschaft ermitteln können. Die `DateTimeKind`-Aufzählung, die der Typ dieser Eigenschaft ist, besitzt die Werte `Local` (lokales Datum), `Utc` (UTC-Datum) und `Unspecified` (nicht spezifizierter Typ).

Eine `DateTime`-Instanz vom Typ `Local` verwaltet intern ihren Offset zum UTC-Datum, wobei auch eine gegebenenfalls zu dem Datum gültige Sommerzeit berücksichtigt wird. Ein unspezifiziertes Datum verwaltet allerdings keinen Offset.

Der Typ des Datums spielt bei der Serialisierung eine große Rolle, denn bei der Deserialisierung wird der Typ natürlich berücksichtigt. Angenommen, das Datum »01.07.2010 12:00:00«

wird auf einem System mit der deutschen Zeitzone nach XML serialisiert, dann erhalten Sie die folgenden XML-Daten für die einzelnen Datumstypen:

- Unspecified: <dateTime>2010-07-01T12:00:00</dateTime>
- Local: <dateTime>2010-07-01T12:00:00+02:00</dateTime>
- Utc: <dateTime>2010-07-01T10:00:00Z</dateTime>

Bei einem unspezifizierten Datum wird keine Information über den UTC-Offset gespeichert. Wird ein solches Datum auf einem System mit einer anderen Zeitzone deserialisiert, resultiert ein falsches Datum, wenn man davon ausgeht, dass es sich eigentlich um ein lokales Datum handelt.

Bei einem lokalen Datum wird der Offset zum UTC-Datum mit serialisiert. Im Beispiel sind das deswegen zwei Stunden, weil zu dem normalen Offset von einer Stunde noch die Sommerzeit hinzukommt (die ja am 1.7. in Deutschland verwendet wird).

Ein UTC-Datum wird mit dem Suffix »Z« serialisiert, der das Datum als UTC-Datum kennzeichnet.

Nun stellt sich noch die Frage, wann ein Datum ein unspezifiziertes, ein lokales oder ein UTC-Datum ist. Ein auf `DateTime.Now` oder `DateTime.Today` basierendes (z. B. über `DateTime.Now.AddDays(1)` erzeugtes) oder über die `ToLocalTime`-Methode erzeugtes `DateTime`-Objekt ist immer vom Typ `Local`. Ein über den `DateTime`-Konstruktor, die `Convert.ToDateTime`, die `DateTime.Parse`- oder eine andere statische `DateTime`-Methode erzeugtes `DateTime`-Objekt ist vom Typ `Unspecified`[4], sofern Sie nicht – wie beim Konstruktor möglich – den Datumstyp explizit angeben. Ein über die `ToUniversalTime`-Methode einer `DateTime`-Instanz erzeugtes `DateTime`-Objekt ist schließlich vom Typ `Utc`. Schließlich können Sie den Typ eines Datums noch über die statische `DateTime`-Methode `SpecifyKind` ändern.

Die Verwendung von `DateTime` ist für den Austausch zwischen verschiedenen Systemen also problematisch:

- Handelt es sich um ein unspezifiziertes Datum, das ja sehr leicht unbedacht zum Beispiel über die `Convert.ToDateTime`-Methode erzeugt werden kann, arbeitet der Empfänger unter Umständen mit einem falschen Datum.
- Handelt es sich um ein lokales oder UTC-Datum, muss der Empfänger den Typ des Datums auf jeden Fall berücksichtigen, bevor er dieses weiterverarbeitet.

Microsoft hat dieses Problem erkannt und deshalb im .NET Framework 3.5 die Klasse `DateTimeOffset` entwickelt. Diese Klasse befindet sich zwar im Namensraum `System`, erfordert aber das Referenzieren der Assembly *Sytem.Core.dll*.

Eine `DateTimeOffset`-Instanz verwaltet *nicht*, wie der Name vermuten lässt, einen Offset zu einen Datum, sondern ein unspezifiziertes Datum *und* dessen Offset zum UTC-Datum[5]. Beim Erzeugen einer `DateTimeOffset`-Instanz übergeben Sie dem Konstruktor einfach ein `DateTime`-Objekt:

4. Bei einem Datum, das nicht aus dem System ausgelesen wurde, kann nicht eindeutig bestimmt werden, ob es sich um ein lokales oder ein UTC-Datum handelt.
5. DateTimeWithOffset wäre eigentlich ein besserer Name gewesen.

```
string input = "1.7.2010 12:00";
DateTime dateTime = Convert.ToDateTime(input);
DateTimeOffset dateTimeOffset = new DateTimeOffset(dateTime);
```

Listing 120: Erzeugen einer DateTimeOffset-Instanz auf der Basis eines konvertierten Strings

Sie können das gespeicherte Datum aus der `DateTime`-Eigenschaft auslesen. `DateTime` verwaltet immer den übergebenen Datumswert. Den Offset zum UTC-Datum erhalten Sie über die `Offset`-Eigenschaft (in Stunden). Übergeben Sie ein lokales oder unspezifiziertes Datum, wird der Offset entsprechend der im System aktuellen Zeitzone[6] gespeichert. Übergeben Sie ein UTC-Datum, speichert `Offset` Null.

Der Empfänger einer serialisierten `DateTimeOffset`-Instanz kann nun einfach die Eigenschaft `LocalDateTime` auslesen, um das für sein System lokale Datum zu ermitteln, oder die Eigenschaft `UtcDateTime`, um das UTC-Datum auszulesen:

```
Console.WriteLine("DateTimeOffset.LocalDateTime: " +
   dateTimeOffset.LocalDateTime.ToString());
Console.WriteLine("DateTimeOffset.UtcDateTime: " +
   dateTimeOffset.UtcDateTime.ToString());
```

Listing 121: Auslesen des für das aktuelle System geltenden lokalen und des UTC-Datums

Hinweis

Das Beispiel zu diesem Rezept ist ein wenig umfangreicher und demonstriert `DateTimeOffset` für je ein unspezifiziertes, ein lokales und ein UTC-Datum. Außerdem finden Sie in dem Beispiel-Ordner einen WCF-Dienst, der die drei unterschiedlichen `DateTime`-Typen und ein `DateTimeOffset`-Objekt liefert. Starten Sie diese Konsolenanwendung, ändern Sie die Zeitzone auf Ihrem System und starten Sie danach die Client-Anwendung (die Sie natürlich auch in dem Ordner des Beispiels finden). Sie werden sehen, dass das Datum über eine `DateTimeOffset`-Instanz immer korrekt und unmissverständlich übergeben wird, was bei den `DateTime`-Varianten nicht der Fall ist (bei unspezifizierten Datumswerten wird ein nicht korrekt interpretierbares Datum übertragen, bei lokalen oder UTC-Datumswerten muss der Client wissen (bzw. abfragen), ob es sich um ein lokales oder UTC-Datum handelt).

78 Mit Zeitzonen arbeiten

Wenn Sie mit Zeitzonen arbeiten wollen (oder müssen ☺), können Sie dazu die Klasse `TimeZoneInfo` aus dem Namensraum `System` verwenden (allerdings müssen Sie dazu die Assembly *System.Core.dll* referenzieren). Diese Klasse erlaubt:

- das Ermitteln von Informationen zur aktuellen Zeitzone,
- den Vergleich von zwei Zeitzonen,

6. Korrekter wäre eigentlich: »der beim Start der Anwendung auf dem System aktuellen Zeitzone«, da .NET-Anwendungen eine Änderung der Zeitzone nach ihrem Start leider nicht berücksichtigen.

- das Einlesen der auf dem System gespeicherten Zeitzonen und
- das Erstellen einer benutzerdefinierten Zeitzone (was in diesem Rezept nicht erläutert wird).

Tabelle 11 fasst zunächst einmal die wichtigen Eigenschaften und Methoden der `TimeZoneInfo`-Klasse zusammen.

Eigenschaft	Beschreibung
`Id`	verwaltet einen String, der die ID der Zeitzone darstellt. Über die statische Methode `FindSystemTimeZoneById` können Sie unter Übergabe einer Zeitzonen-ID eine bestimmte Zeitzone ermitteln.
`DisplayName`	gibt den englischen Anzeige-Namen der Zeitzone zurück
`DaylightName`	gibt den englischen Namen der Sommerzeit zurück
`BaseUtcOffset`	liefert einen `TimeSpan`-Wert mit dem Offset zur UTC-Zeit ohne Berücksichtigung einer eventuell gesetzten Sommerzeit
`SupportsDaylightSavingTime`	gibt an, ob die Zeitzone eine Sommerzeit besitzt

`TimeSpan GetUtcOffset(DateTime` ***dateTime***`)`
 	
`TimeSpan GetUtcOffset(DateTimeOffset` ***dateTimeOffset***`)`
	liefert den Offset zur UTC-Zeit, der an dem übergebenen Datum gültig war bzw. ist. Die Übergabe des Datums ist deswegen wichtig, weil diese Methode auch die Sommerzeit berücksichtigt. Bei der Übergabe des 1.1.2008 wird für die in Deutschland lokale Zeitzone zum Beispiel eine Stunde Offset zurückgegeben. Bei der Übergabe des 1.7.2008 werden zwei Stunden zurückgegeben, da an diesem Datum die Sommerzeit gilt. Über `GetUtcOffset(DateTime.Now)` erhalten Sie den aktuellen Offset zur UTC-Zeit.

`bool IsDaylightSavingTime(DateTime` ***dateTime***`)`
 	
`bool IsDaylightSavingTime(DateTimeOffset` ***dateTimeOffset***`)`
	gibt an, ob es sich bei den übergebenem Datum um ein Datum mit Sommerzeit handelt

`bool HasSameRules (TimeZoneInfo` ***other***`)`	ermittelt, ob die übergebene Zeitzone dieselben Regeln (bezüglich des UTC-Offsets und der Sommerzeit) besitzt

Tabelle 11: Die wichtigen Eigenschaften der `TimeZoneInfo`*-Klasse*

Die folgenden Beispiele erfordern die Referenzierung der Assembly *System.Core.dll* und das Importieren der Namensräume `System` und `System.Collections.ObjectModel`.

Eine `TimeZoneInfo`-Instanz für die auf dem jeweiligen System aktuelle Zeitzone erhalten Sie über die statische Eigenschaft `Local`:

```
TimeZoneInfo localTimeZoneInfo = TimeZoneInfo.Local;
Console.WriteLine(
```

Listing 122: Ermittlung von Informationen zur lokalen Zeitzone

```
    "Aktuelle Zeitzone:" + Environment.NewLine +
    "Anzeige-Name: " + localTimeZoneInfo.DisplayName + Environment.NewLine +
    "Sommerzeit-Name: " + localTimeZoneInfo.DaylightName +
    Environment.NewLine +
    "Basis-UTC-Offset (ohne Sommerzeit): " + localTimeZoneInfo.BaseUtcOffset +
    Environment.NewLine +
    "Zurzeit Sommerzeit: " +
    localTimeZoneInfo.IsDaylightSavingTime(DateTime.Now) +
    Environment.NewLine +
    "Sommerzeit wird unterstützt: " +
    localTimeZoneInfo.SupportsDaylightSavingTime + Environment.NewLine +
    "Aktueller UTC-Offset (inkl. Sommerzeit): " +
    localTimeZoneInfo.GetUtcOffset(DateTime.Now));
```

Listing 122: Ermittlung von Informationen zur lokalen Zeitzone (Forts.)

Achtung

`TimeZoneInfo.Local` gibt immer die beim Start der Anwendung aktuelle Zeitzone zurück. Wird die Zeitzone nach dem Start der Anwendung geändert, wird `TimeZoneInfo.Local` nicht aktualisiert.

Eine bestimmte Zeitzone erhalten Sie über die statische Methode `FindSystemTimeZoneById`. Aus dieser Methode übergeben Sie die String-Id der zu ermittelnden Zeitzone. Das folgende Beispiel ermittelt die Zeitzone für Neuseeland:

```
TimeZoneInfo newZealandTimeZoneInfo = TimeZoneInfo.FindSystemTimeZoneById(
   "New Zealand Standard Time");
```

Hinweis

Die Ids der auf dem System verfügbaren Zeitzonen werden in der Registry verwaltet und können sich deswegen auf verschiedenen Systemen voneinander unterscheiden. Deswegen ist es schwierig zu sagen, ob die auf einem System gültige Id einer Zeitzone auf einem anderen System existiert. Wird die Id nicht gefunden, resultiert eine `TimeZoneNotFoundException`. Die Ids der auf dem aktuellen System verfügbaren Zeitzonen können Sie ermitteln, indem Sie die einzelnen Zeitzonen durchgehen, wie ich es im letzten Beispiel dieses Rezepts beschreibe.

Den Offset einer Zeitzone zu einer anderen können Sie berechnen, indem Sie den UTC-Offset der einen Zeitzone vom UTC-Offset der anderen Zeitzone abziehen:

```
DateTime now = DateTime.Now;
TimeSpan offsetToLocalTimeZone = newZealandTimeZoneInfo.GetUtcOffset(now) -
   TimeZoneInfo.Local.GetUtcOffset(now);
```

Schließlich (aber nicht endlich ☺) können Sie noch die auf dem System verfügbaren Zeitzonen durchgehen:

```
ReadOnlyCollection<TimeZoneInfo> systemTimeZones =
   TimeZoneInfo.GetSystemTimeZones();
foreach (TimeZoneInfo timeZoneInfo in systemTimeZones)
{
   Console.WriteLine(timeZoneInfo.DisplayName);
}
```

Listing 123: Durchgehen der auf dem aktuellen System verwalteten Zeitzonen

79 Mit Kalendersystemen arbeiten

Zur Einteilung eines Jahres in Monate und Tage existieren eine Menge Kalendersysteme, von denen die meisten allerdings mittlerweile veraltet sind. Die in Deutschland und den meisten anderen Ländern gebräuchliche Zeitrechnung basiert auf dem so genannten gregorianischen Kalender. Dieser von Papst Gregor XIII im 16. Jahrhundert entwickelte Kalender verwendet das Geburtsjahr von Jesus Christus als Basis und teilt das Jahr in durchschnittlich 365,2425 Tage ein. Aufgrund des Nachkommanteils existieren Schaltjahre, in denen der Februar einen zusätzlichen Tag besitzt.

Exkurs

Ein Jahr ist immer dann ein Schaltjahr,

- wenn die Jahreszahl durch vier ohne Rest teilbar ist und
- wenn die Jahreszahl nicht durch 100 ohne Rest teilbar ist oder wenn die Jahreszahl durch 400 ohne Rest teilbar ist.

Schaltjahre gibt es also alle vier Jahre, alle 100 Jahre nicht, aber alle 400 Jahre dann doch wieder.

Hinweis

Interessant beim gregorianischen Kalender ist noch, dass dieser Kalender laut Wikipedia (*www.wikipedia.de*) in 14 Jahreskalender unterteilt wird, die an verschiedenen Wochentagen beginnen und scheinbar auch keine Schaltjahre besitzen können. Dazu ist im Internet aber leider nicht allzu viel zu finden.

Neben dem gregorianischen Kalender existiert eine Vielzahl anderer Kalendersysteme. Viele davon sind mittlerweile veraltet. Wenn Sie sich nun fragen, warum Sie sich mit Kalendersystemen auseinandersetzen sollten, sollten Sie beachten, dass einige Kalendersysteme auch heute noch in Gebrauch sind. Der japanische Kalender wird z. B. in Japan eingesetzt (wo auch sonst ...). Wenn Sie also für Länder Anwendungen entwickeln, die einen anderen Kalender einsetzen als den gregorianischen, oder wenn Ihre Anwendungen mit Anwendungen in diesen Ländern kommunizieren, kann es schon einmal vorkommen, dass Sie Datumswerte in einem anderen Kalendersystem verarbeiten müssen.

Die meisten Kalender sind Solarkalender und basieren auf dem Sonnenjahr. Lunarkalender (wie z. B. der islamische Kalender) hingegen basieren auf einem Mondjahr, das in 12 Mondmonate unterteilt wird. Die sich daraus ergebende Jahreslänge ist um ca. 10 bis 12 Tage kürzer als ein Sonnenjahr. Um diese massive Verschiebung auszugleichen besitzen Lunarkalender Korrekturen, die in der Regel einigen Jahren einen zusätzlichen Monat hinzufügen. Schließlich

existieren mit den Lunisolarkalendern noch Mischformen, bei denen das Jahr nach der Sonne, die Monate und Tage aber nach dem Mond berechnet werden.

Die Jahreszahl eines Kalenders basiert auf einer so genannten Ära. Die christliche Ära, auf der z. B. der gregorianische Kalender basiert, beginnt mit der Geburt von Jesus Christus. Die Ära, auf der der thailändische buddhistische Kalender basiert, beginnt mit der Geburt von Buddha, die 543 Jahre früher war. So kommt es, dass einige Kalender zwar denselben Grund-Algorithmus verwenden, die Jahreszahl aber eine andere ist. Das gregorianische Jahr 2010 entspricht z. B. dem thailändischen Jahr 2553.

Die meisten modernen Kalender besitzen lediglich eine Ära. Der japanische Kalender besitzt allerdings mehrere (zurzeit vier), weil in Japan jedes Mal, wenn ein neuer Kaiser sein Amt antritt, eine neue Ära beginnt. Da der aktuelle Kaiser Heisei 1989 sein Amt angetreten hat, entspricht das gregorianische Jahr 1989 dem japanischen Jahr 1 in der Ära Heisei. 2010 entspricht demnach dem japanischen Jahr 22 in der Ära Heisei.

Die wichtigsten Kalendersysteme werden über Klassen des Namensraums `System.Globalization` repräsentiert (Tabelle 12).

Kalendersystem	Beschreibung	.NET-Klasse	Aktuelle Verwendung
Julianischer Kalender	Einer der ersten Kalender. Wurde von Julius Cäsar entwickelt und war in einigen Ländern noch bis in das 20. Jahrhundert gültig.	`JulianCalendar`	Nur noch bei einigen orthodoxen Kirchen und in den Geschichtswissenschaften gebräuchlich.
Gregorianischer Kalender	Der zurzeit in den meisten Ländern verwendete Kalender (*de.wikipedia.org/wiki/Gregorianischer_Kalender*).	`GregorianCalendar`	Wird in den meisten Ländern verwendet.
Chinesischer Kalender	Alter chinesischer Kalender (China benutzt heute den gregorianischen Kalender).	`ChineseLunisolarCalendar`	Wird in China zur Bestimmung traditioneller Feste verwendet.
Hebräischer bzw. jüdischer Kalender	Lunisolarkalender mit einem Extra-Monat in einigen Jahren, der die knapp 11 Tage Unterschied zwischen 12 Mond-Monaten und einem Sonnen-Jahr ausgleicht (*en.wikipedia.org/wiki/Hebrew_calendar*).	`HebrewCalendar`	Wird im Judentum verwendet.
Islamischer Kalender (Hijri-Kalender)	Alter Kalender, der in den meisten islamischen Ländern in Gebrauch war (*en.wikipedia.org/wiki/Islamic_calendar*).	`HijriCalendar`	Ist heute weitestgehend durch den gregorianischen Kalender ersetzt, wird aber von Muslimen in aller Welt (außer in Saudi-arabien) zur Bestimmung von islamischen Feiertagen verwendet.

Tabelle 12: Die im .NET Framework abgebildeten Kalendersysteme

Kalendersystem	Beschreibung	.NET-Klasse	Aktuelle Verwendung
Japanischer Kalender	Auf dem gregorianischen Kalender basierender Kalender. Im Unterschied zum gregorianischen Kalender werden die Jahre in einzelne Ären eingeteilt, wobei jede Ära mit dem Amtsantritt eines neuen Kaisers beginnt und am 31.12. des Jahres seines Ausscheidens endet. Das Jahr 2008 entspricht demnach dem Jahr 20 in der Ära »Heisei«, da der Kaiser Heisei 1989 sein Amt angetreten hat (*de.wikipedia.org/wiki/Japanischer_Kalender*).	`JapaneseCalendar`	Wird in Japan eingesetzt.
Japanischer Lunisolar-Kalender	Alter, in Japan gebräuchlicher Kalender, der im Wesentlichen dem chinesischen Lunisolarkalender entspricht.	`JapaneseLunisolarCalendar`	Wird nicht mehr verwendet (wurde 1873 durch den japanischen Kalender ersetzt).
Koreanischer Kalender	Basiert im Wesentlichen auf dem gregorianischen Kalender, allerdings werden die Ära und damit auch die Jahreszahl anders gerechnet (*en.wikipedia.org/wiki/Korean_calendar*).	`KoreanCalendar`	Wird in (Nord?)Korea eingesetzt.
Koreanischer Lunisolar-Kalender	Alter, in Korea verwendeter Kalender	`KoreanLunisolarCalendar`	Wird in Korea zur Bestimmung von Feiertagen eingesetzt.
Iranischer bzw. Persischer Kalender	Solarkalender mit einem 365-Tage-Jahr. Jeder Monat besitzt 30 Tage. 5 zusätzliche Tage werden zwischen dem achten und dem neunten Monat eingefügt. Schaltjahre werden mit einer recht komplizierten Arithmetik berechnet (*www.ortelius.de/kalender/pers_en.php*).	`PersianCalendar`	Wird im Iran und in Afghanistan eingesetzt.
Thailändischer Buddhistischer Kalender	Sonnenbasierter Kalender, der auf dem gregorianischen Kalender basiert. Die Jahreszahl wird allerdings anders basierend auf der Buddhistischen Ära berechnet, die 543 Jahre früher beginnt als die christliche (*en.wikipedia.org/wiki/Thai_solar_calendar*).	`ThaiBuddhistCalendar`	Wird offiziell in Thailand eingesetzt (in der Geschäftswelt wird allerdings häufig der gregorianische Kalender eingesetzt)

Tabelle 12: Die im .NET Framework abgebildeten Kalendersysteme (Forts.)

Kalendersystem	Beschreibung	.NET-Klasse	Aktuelle Verwendung
Taiwanesischer Kalender	Basiert auf dem gregorianischen Kalender, allerdings beginnt die moderne Ära mit der Gründung der Republik China im Jahr 1912 (*de.wikipedia.org/wiki/Chinesischer_Kalender*).	`TaiwanCalendar`	Wird in Taiwan eingesetzt.
Taiwanesischer Lunisolarkalender	Lunisolarkalender, der die Jahre wie der gregorianische Kalender berechnet. Die Berechnung der Monate und Tage basiert aber auf dem Mond.	`TaiwanLunisolar-Calendar`	Wird wahrscheinlich[7] in Taiwan zur Bestimmung traditioneller Feiertage eingesetzt (*www.gio.gov.tw/info/festival_c/index_e.htm*)
Saudiarabisch islamischer bzw. Umm-Al-Qura-Kalender	Der Umm-Al-Qura-Kalender[8] ist der islamische Lunarkalender, der in Saudiarabien eingesetzt wird. (*en.wikipedia.org/wiki/Islamic_calenda*, *www.phys.uu.nl/~vgent/islam/ummalqura.htm*)	`UmAlQuraCalendar`	Wird in Saudiarabien wahrscheinlich zur Bestimmung von traditionellen islamischen Feiertagen verwendet (*www.phys.uu.nl/~vgent/islam/ummalqura.htm*)

Tabelle 12: Die im .NET Framework abgebildeten Kalendersysteme (Forts.)

Hinweis

Wie Sie der Tabelle entnehmen können, ist es im Einzelfall nicht einfach, herauszufinden, wofür ein Kalender in der heutigen Zeit (noch) eingesetzt wird. Wenn Sie Anwendungen für Nicht-europäische Länder entwickeln, sollten Sie einfach nachfragen, ob Ihre Kunden einen anderen als den gregorianischen Kalender verwenden.

Zuordnung von Kalendersystemen zu den Kulturen

Die `CultureInfo`-Klasse besitzt die Eigenschaft `Calendar`, über die Sie den in der jeweiligen Kultur verwendeten Haupt-Kalender ermitteln können. Über die Array-Eigenschaft `Optional-Calendars` erhalten Sie eine Auflistung aller Kalender, die die Kultur unterstützt, wobei der Hauptkalender nach meinen Erfahrungen nicht immer an erster Stelle steht. `OptionalCalendars` referenziert Instanzen vom Typ `Calendar`, der der Basistyp aller Kalender-Klassen ist.

So können Sie z. B. den Haupt- und die optionalen Kalender der aktuellen Kultur ermitteln:

```
// Aktuellen Haupt-Kalender ermitteln
Calendar currentCalendar = Thread.CurrentThread.CurrentCulture.Calendar;
Console.WriteLine("Hauptkalender der aktuellen Kultur:");
Console.WriteLine(currentCalendar.ToString());
```

Listing 124: Ermitteln des Hauptkalenders und der optionalen Kalender der aktuellen Kultur

7. Wo der taiwanesische Lunisolarkalender eingesetzt wird, ist schwer herauszufinden. Im Internet ist lediglich die Rede von einem Lunarkalender *(www.gio.gov.tw/info/festival_c/index_e.htm)*. Außerdem referenziert keine CultureInfo-Instanz für die im .NET Framework vorhandenen spezifischen Kulturen diesen Kalender als Haupt- oder optionalen Kalender.
8. Der Name dieses Kalenders ist tatsächlich »Umm Al-Qura«. Microsoft hat die Klasse wohl falsch benannt.

```
// Die optionalen Kalender der aktuellen Kultur ermitteln
Console.WriteLine();
Console.WriteLine("Optionale Kalender:");
foreach (Calendar calendar in
   Thread.CurrentThread.CurrentCulture.OptionalCalendars)
{
   Console.WriteLine(calendar.ToString());
}
```

Listing 124: Ermitteln des Hauptkalenders und der optionalen Kalender der aktuellen Kultur

Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading` und `System.Globalization`.

Ermitteln von Informationen zu einem Kalender

Die `Calendar`-Klasse, die die Basis für alle Kalender-Klassen ist, bietet einige Eigenschaften und Methoden zur Arbeit mit einem Kalender. Einige wichtige, wie zum Beispiel der Name des Kalenders, fehlen aber eigenartigerweise. Das folgende Listing fasst die wichtigsten Möglichkeiten am Beispiel des japanischen Kalenders zusammen:

```
// Instanz der JapaneseCalendar-Klasse erzeugen
Calendar japaneseCalendar = new JapaneseCalendar();

Console.WriteLine();
Console.WriteLine("Japanischer Kalender:");

// Name ermitteln (leider wird der Name
// nicht über eine Eigenschaft zur Verfügung gestellt)
string calendarName = japaneseCalendar.ToString();
if (calendarName.StartsWith("System.Globalization"))
{
   calendarName = calendarName.Substring(21, calendarName.Length - 29);
}
Console.WriteLine("Name: " + calendarName);

// Den Algorithmus-Typ ermitteln
Console.Write("Typ: ");
switch (japaneseCalendar.AlgorithmType)
{
   case CalendarAlgorithmType.LunarCalendar:
      Console.WriteLine("Lunarkalender");
      break;

   case CalendarAlgorithmType.LunisolarCalendar:
      Console.WriteLine("Lunisolarkalender");
```

Listing 125: Ermittlung der wichtigsten Informationen für einen Kalender

```
      break;

   case CalendarAlgorithmType.SolarCalendar:
      Console.WriteLine("Solarkalender");
      break;

   case CalendarAlgorithmType.Unknown:
      Console.WriteLine("Unbekannt");
      break;
}

// Die aktuelle Ära ermitteln
Console.WriteLine("Aktuelle Ära:" + japaneseCalendar.GetEra(DateTime.Now));

// Die verfügbaren Ären ermitteln
string availableEras = null;
foreach (int era in japaneseCalendar.Eras)
{
   if (availableEras != null)
   {
      availableEras += ", ";
   }
   availableEras += era.ToString();
}
Console.WriteLine("Verfügbare Ären: " + availableEras);

// Einen String ermitteln, der das aktuelle Datum
// für den japanischen Kalender darstellt
string japaneseDateString =
   japaneseCalendar.GetDayOfMonth(DateTime.Now) + "." +
   japaneseCalendar.GetMonth(DateTime.Now) + "." +
   japaneseCalendar.GetYear(DateTime.Now) + " " +
   "in der Ära " + japaneseCalendar.GetEra(DateTime.Now);
```

Listing 125: Ermittlung der wichtigsten Informationen für einen Kalender (Forts.)

Konvertieren vom aktuellen in ein anderes Kalendersystem

Wenn Sie einen String erzeugen müssen, der ein Datum in einer anderen Kultur enthält, können Sie einfach der `ToString`-Methode eines `DateTime`-Objekts ein entsprechendes `CultureInfo`-Objekt übergeben. Beim Formatieren des Datumswerts wird per Voreinstellung das Haupt-Kalendersystem der Kultur verwendet. Wollen Sie ein anderes Kalendersystem verwenden, können Sie der `Calendar`-Eigenschaft des `DateTimeFormatInfo`-Objekts ein entsprechendes `Calendar`-Objekt zuweisen, das über die `DateTimeFormat`-Eigenschaft erreichbar ist.

Das folgende Beispiel erzeugt einen Datums-String in der taiwanesischen Kultur mit dem Taiwan-Kalender (der nicht der Haupt-Kalender dieser Kultur ist):

```
CultureInfo taiwanCulture = new CultureInfo("zh-TW");
taiwanCulture.DateTimeFormat.Calendar = new TaiwanCalendar();
string taiwanDateString = DateTime.Now.ToString(taiwanCulture);
```

Listing 126: Erzeugen eines Datums-Strings für die taiwanesische Kultur und den Taiwan-Kalender

Über verschiedene Get-Methoden eines Kalenders, denen Sie jeweils ein Datum übergeben, können Sie die für den Kalender gültigen Datums-Teile, wie das Jahr, den Monat und den Tag ermitteln. So können Sie ein Datum für einen bestimmten Kalender auch über die einzelnen Teile des Datums darstellen:

```
UmAlQuraCalendar ummAlQuraCalendar = new UmAlQuraCalendar();
int ummAlQuraYear = ummAlQuraCalendar.GetYear(DateTime.Now);
int ummAlQuraMonth = ummAlQuraCalendar.GetMonth(DateTime.Now);
int ummAlQuraDay = ummAlQuraCalendar.GetDayOfMonth(DateTime.Now);
```

Listing 127: Ermitteln des Jahres, des Monats und des Tags für den Umm-Al-Qura-Kalender

Konvertieren von anderen in das aktuelle Kalendersystem

Wenn Sie die einzelnen Teile eines Datums (Tag, Monat, Jahr etc.) in einem anderen Kalendersystem erhalten, können Sie einfach den entsprechenden Kalender instanzieren und dessen `ToDateTime`-Methode aufrufen. Dieser Methode übergeben Sie `int`-Werte für das Jahr, den Monat, den Tag, die Stunde etc.

Das folgende Listing konvertiert die Werte aus dem obigen Beispiel in ein Datum für die aktuelle Kultur:

```
DateTime date = ummAlQuraCalendar.ToDateTime(ummAlQuraYear,
   ummAlQuraMonth, ummAlQuraDay, 0, 0, 0, 0);
```

Listing 128: Konvertieren von Datums-Einzelwerten von einem Kalendersystem in das Kalendersystem der aktuellen Kultur

Wenn Sie einen Datums-String erhalten, liegt dieser zum einen möglicherweise in dem Datumsformat vor, das der Kultur entspricht, der das Datum entstammt. Zum anderen liegt das Datum unter Umständen in einem anderen als dem aktuellen Kalendersystem vor. Zur Konvertierung des Strings müssen Sie also wissen, welcher Kultur der String entstammt und welches Kalendersystem verwendet wurde. Dann können Sie einfach eine `CultureInfo`-Instanz für die entsprechende Kultur erzeugen und diesem Objekt den für Formatierungen verwendeten Kalender zuweisen. Dazu verwenden Sie allerdings nicht die `Calendar`-Eigenschaft (da diese schreibgeschützt ist), sondern die `Calendar`-Eigenschaft des `DateTimeFormatInfo`-Objekts, das Sie über die `DateTimeFormat`-Eigenschaft erreichen:

```
// Ein Taiwan-Datum in ein Datum in der aktuellen
// Kultur konvertieren
CultureInfo convertCulture = new CultureInfo("zh-TW");
convertCulture.DateTimeFormat.Calendar = new TaiwanCalendar();
date = Convert.ToDateTime(taiwanDateString, convertCulture);
Console.WriteLine("Umgewandeltes Datum: " + date.ToShortDateString());
```

Listing 129: Konvertieren eines Datums-Strings, der in einer bestimmten Kultur und einem bestimmten Kalendersystem vorliegt, in ein Datum für die aktuelle Kultur

Das zum Zeitpunkt des Schreibens dieses Artikels aktuelle taiwanesische Datum »96/7/19« wird auf einem europäischen System zum Beispiel in das Datum 19.7.2007 umgewandelt.

Die Klasse CalendarInfo

Da die `Calendar`-Klasse einige Informationen über einen Kalender vermissen lässt, habe ich die Klasse `CalendarInfo` entwickelt. Diese Klasse liefert unter anderem einen Namen für den dem Konstruktor übergebenen Kalender und Informationen über die minimale und die maximale Anzahl von Tagen im Monat. Sie finden diese Klasse im Repository und in der Codebook-Klassenbibliothek. Außerdem habe ich neben dem Basis-Beispiel für dieses Rezept noch ein Beispiel mit einem Windows-Formular implementiert, das die `CalendarInfo`-Klasse einsetzt.

Anwendungen, Anwendungs-Konfiguration, Prozesse und Dienste

80 Den Dateinamen der Anwendung ermitteln

Den vollen Dateinamen der ausführenden Anwendung (inklusive Pfad) erhalten Sie in einer Windows-Anwendung über die Eigenschaft `ExecutablePath` des `Application`-Objekts:

```
string filename = System.Windows.Forms.Application.ExecutablePath;
```

Um die `Application`-Klasse verfügbar zu haben, benötigt die Anwendung einen Verweis auf die Assembly *System.Windows.Forms.dll*, was ohne Probleme auch in einer Konsolenanwendung oder Klassenbibliothek möglich ist.

Ohne diesen Verweis können Sie aber auch stattdessen die Eigenschaft `Location` eines `Assembly`-Objekts abfragen, das die »Eintritts-Assembly« (das ist die Assembly, über die das Programm gestartet wurde) repräsentiert. Dieses Objekt erhalten Sie über die Methode `GetEntryAssembly` der `Assembly`-Klasse, die Bestandteil des `System.Reflection`-Namensraums ist:

```
string filename = System.Reflection.Assembly.GetEntryAssembly().Location;
```

Hinweis

Als *Reflektion* (Reflection) werden Features bezeichnet, über die Sie in der Laufzeit eines Programms Informationen über dessen Assemblys, die darin enthaltenen Klassen und deren Schnittstellen, Methoden, Eigenschaften und Ereignisse etc. auslesen können. Ein weiterer Teil von Reflektion ist das dynamische Erzeugen von Klassen und Assemblys, also das Erzeugen von Programmcode in der Laufzeit eines Programms.

Häufig wird auch vorgeschlagen, die »ausführende Assembly« zu verwenden, die Sie über die Methode `GetExecutingAssembly` erreichen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Dabei müssen Sie allerdings aufpassen: Wird diese Methode in einer Klassenbibliothek aufgerufen, liefert sie ein `Assembly`-Objekt zurück, das die Assembly der *Klassenbibliothek* repräsentiert. Der ausgelesene Dateiname ist also in diesem Fall nicht der der Anwendung, sondern der der Klassenbibliothek.

In allen Fällen erhalten Sie den Dateinamen inklusive Pfad. Wenn Sie nur den Namen der Datei auslesen wollen, können Sie diesen über ein `FileInfo`-Objekt ermitteln. Ein solches Objekt dient dem Auslesen von Informationen zu einer Datei und liefert u. a. über die Eigenschaft `Name` den Dateinamen. Die folgende Funktion liest den Dateinamen der Anwendung aus und liefert diesen zurück:

```
public static string GetApplicationFilename()
{
   // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-
   // Assembly speichert
   System.IO.FileInfo fi = new System.IO.FileInfo(
      System.Reflection.Assembly.GetEntryAssembly().Location);
   // Dateiname auslesen und zurückgeben
   return fi.Name;
}
```

Listing 130: Methode zum Ermitteln des Dateinamens der Anwendung

81 Das Verzeichnis der Anwendung ermitteln

In einer Anwendung mit Referenz auf die Assembly *System.Windows.Forms.dll* können Sie den Pfad der Anwendung über die Eigenschaft `StartupPath` des `Application`-Objekts ermitteln:

```
string applicationPath = System.Windows.Forms.Application.StartupPath;
```

Ohne Referenz auf die *System.Windows.Forms.dll* können Sie stattdessen auch den vollen Dateinamen der »Eintritts-Assembly« ermitteln und über ein `FileInfo`-Objekt den Pfad auslesen:

```
public static string GetApplicationFolderName()
{
   // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-
   // Assembly speichert */
   System.IO.FileInfo fi = new System.IO.FileInfo(
      System.Reflection.Assembly.GetEntryAssembly().Location);

   // Den Pfad des Verzeichnisses der Datei zurückgeben
   return fi.DirectoryName;
}
```

Listing 131: Methode zur Ermittlung des Ordners, in dem die Anwendung gespeichert ist

Beide Varianten funktionieren auch ohne Probleme in einer Klassenbibliothek. Sie erhalten dann allerdings das Verzeichnis der Anwendung zurück, nicht das der Klassenbibliothek.

82 Befehlszeilenargumente auswerten

Viele Standardanwendungen können mit Befehlszeilenargumenten aufgerufen werden. Dem Windows-Explorer können Sie zum Beispiel beim Aufruf den Pfad zu einem Ordner übergeben, den dieser anzeigen soll:

```
explorer C:\Windows
```

Befehlszeilenargumente werden dabei durch Leerzeichen vom Programmdateinamen und von anderen Befehlszeilenargumenten getrennt. Beim folgenden Aufruf eines Programms:

```
Demo.exe Das ist ein Test
```

werden die Argumente "Das", "ist", "ein" und "Test" übergeben. Werden Argumente in Anführungszeichen eingeschlossen, resultiert ein einziges Argument:

```
Demo.exe "Das ist ein Test"
```

Dieses Beispiel resultiert in dem einzigen Argument "Das ist ein Test". Argumente, die selbst Leerzeichen enthalten, müssen beim Aufruf also in Anführungszeichen eingeschlossen werden:

```
explorer "C:\Dokumente und Einstellungen"
```

In Ihren (Windows-)Anwendungen können Sie Befehlszeilenargumente ebenfalls auswerten. In einer `Windows.Forms`- oder Konsolenanwendung können Sie dazu die `Main`-Methode, die normalerweise in der Klasse `Program` implementiert ist, um ein Argument vom Typ `string`-Array erweitern:

```
[STAThread]
static void Main(string[] arguments)
{
```

Listing 132: Erweitern der Main-Methode um ein String-Array-Argument

In diesem Array werden alle Argumente übergeben, die beim Aufruf des Programms angegeben wurden. Dieses Vorgehen ist jedoch nicht zu empfehlen, da Sie zum einen auf die Auswertung in der `Main`-Methode eingeschränkt sind. Zum anderen wird die `Main`-Methode in einer mit Visual Studio entwickelten WPF-Anwendung automatisch erzeugt und Sie haben keine Möglichkeit, darin zu programmieren.

Der flexiblere Weg ist der über die `GetCommandLineArgs`-Methode der `Environment`-Klasse aus dem Namensraum `System`. Diese Methode liefert ein `String`-Array zurück, das (anders als der Name vermuten lässt) den kompletten Aufruf der Anwendung enthält. Im ersten Element steht immer der Dateiname der Anwendung. In den folgenden Elementen werden alle Befehlszeilenargumente verwaltet. Damit ist es recht einfach, die einzelnen Argumente auszulesen.

Um keine Probleme bei der Auswertung zu haben, sollten Sie die Syntax der einzelnen Argumente so festlegen, dass diese prinzipiell keine Leerzeichen enthalten. In der Praxis bestehen viele Befehlszeilenargumente aus einem Binde- oder Schrägstrich, gefolgt von einem Namen, einem Doppelpunkt und einem Wert:

```
-Argumentname:Argumentwert
```

Beim Aufruf der Anwendung kann der Argumentwert in Anführungszeichen eingeschlossen werden, sofern dieser Leerzeichen enthält:

```
Imager.exe -imageFolder:"C:\Bilder für die Website"
```

Die Auswertung solcher Argumente ist dann relativ einfach. Dazu gehen Sie die Elemente des von `GetCommandLineArgs` zurückgegebenen `String`-Array ab dem Index 1 durch und vergleichen die einzelnen Argumente mit den erwarteten. Die Programmierung kann an beliebiger Stelle innerhalb der Anwendung erfolgen, wird aber meistens in der `Load`-Methode des Start-Formulars bzw. -Fensters oder in der `Main`-Methode untergebracht.

Das folgende Beispiel liest auf diese Weise die erwarteten Argumente *-debugmode* und *-imagefolder:Ordnerangabe* aus. Beim *imagefolder*-Argument wird die Angabe eines Ordners erwartet, der mit einem Doppelpunkt vom Argumentnamen getrennt wird. Um die Groß-/Kleinschreibung nicht zu berücksichtigen werden die Argumente über die `ToLower`-Methode in Kleinschreibung umgewandelt. Zur Sicherheit werden unbekannte Argumente in der `String`-Variablen `unknownArguments` gesammelt und falls vorhanden nach der Auswertung in einer `MessageBox` gemeldet.

Das Beispiel basiert auf einer WPF-Anwendung mit den üblichen Referenzen und `using`-Direktiven. Im Formular ist ein Label angelegt, das *lblArguments* heißt. In diesem Label werden die übergebenen Befehlszeilenargumente ausgegeben.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
   // Auswerten der Befehlszeilenargumente
   bool debugMode = false;
   string imageFolder = null;
   string unknownArguments = null;
   string[] args = Environment.GetCommandLineArgs();
   for (int i = 1; i < args.Length; i++)
   {
      string loweredArgument = args[i].ToLower();
      if (loweredArgument == "-debugmode")
      {
         debugMode = true;
      }
      else if (loweredArgument.StartsWith("-imagefolder:"))
      {
         // Den Argumentwert auslesen
         imageFolder = args[i].Substring(13, args[i].Length - 13);
      }
      else
      {
         // Unbekanntes Argument
         if (unknownArguments != null)
         {
            unknownArguments += ", ";
         }
         unknownArguments += args[i];
      }
   }

   // Unbekannte Argumente auswerten
   if (unknownArguments != null)
   {
      MessageBox.Show("Die folgenden Argumente sind ungültig: " +
```

Listing 133: Auswerten von Befehlszeilenargumenten

```
            unknownArguments, "Befehlszeilenargumente auswerten",
            MessageBoxButton.OK, MessageBoxImage.Exclamation);
        }

        // Die Argumente auswerten
        this.lblArguments.Content = "debugmode: " + debugMode +
            Environment.NewLine + "imageFolder: " + imageFolder;
    }
```


Listing 133: Auswerten von Befehlszeilenargumenten (Forts.)

Sehr nett von Windows ist, dass in Anführungszeichen eingeschlossene Argumente automatisch so ausgewertet werden, dass die Anführungszeichen entfernt werden. Beim Aufruf mit den Argumenten

```
-debugMode -imageFolder:"C:\Bilder für die Website"
```

werden zum Beispiel "-debugMode" und "-imageFolder:C:\Bilder für die Website" übergeben.

Hinweis

Zum Testen von Befehlszeilenargumenten können Sie diese in Visual Studio in den Eigenschaften des Projekts im Register DEBUGGEN in das Feld BEFEHLSZEILENARGUMENTE eintragen.

83 User Account Control (UAC) berücksichtigen

Ihre Anwendungen sollten die Windows-Vista-Technologie *User Account Control* (UAC) berücksichtigen. Das ist schon deswegen notwendig, weil unter UAC per Voreinstellung alle Anwendungen grundsätzlich mit den eingeschränkten Rechten eines Standard-Benutzers ausgeführt werden, auch wenn der ausführende Benutzer ein Administrator ist. Ein Standard-Benutzer besitzt normalerweise nur eingeschränkte Rechte und kann bestimmte Aktionen, wie z. B. das Einstellen der Systemzeit, nicht ausführen.

Dieses Rezept klärt, wie Sie mit UAC umgehen sollten (ohne allerdings über den Sinn oder Unsinn von UAC zu diskutieren).

Exkurs

User Account Control ist eine Sicherheits-Technologie unter Windows-Vista, die dabei helfen soll, ein System abzusichern. UAC ist per Voreinstellung eingeschaltet, kann aber auch komplett abgeschaltet oder speziell konfiguriert werden. Das Prinzip von UAC basiert darauf, dass Anwendungen normalerweise mit den (eingeschränkten) Rechten ausgeführt werden, die ein Windows-Standardbenutzer besitzt. Das gilt auch dann, wenn der Windows-Benutzer ein Administrator ist. Dies wird dadurch erreicht, dass bereits der Windows-Explorer (der unter anderem auch den Desktop ausführt) mit einem Sicherheits-Token ausgeführt wird, das der Gruppe der Standardbenutzer zugeordnet ist. Jede Anwendung, die unter dem Explorer gestartet wird (was normalerweise jede gestartete Anwendung ist), erbt die Sicherheits-Einstellungen vom Explorer und wird damit auch unter dem eingeschränkten Sicherheits-Token ausgeführt.

Benötigt eine Anwendung nun zur Ausführung einer Aktion Administrationsrechte, fragt UAC in einer speziellen Sicherheits-Box ab, ob zur Ausführung der Aktion die Rechte der Anwendung angehoben werden sollen. Wie die Abfrage aussieht, hängt davon ab, ob der in Windows eingeloggte Benutzer Administrator ist. Ist der Benutzer Administrator, handelt es sich nur um eine einfache Abfrage. Ist der Benutzer kein Administrator, beinhaltet die Sicherheits-Abfrage einen Login, mit dem der Benutzer sich als Administrator authentifizieren kann.

Hinweis

Ausführlichere Informationen zu UAC finden Sie in dem Microsoft-Artikel »Windows Vista Application Development Requirements for User Account Control Compatibility«, den Sie an der Adresse *www.microsoft.com/downloads/details.aspx?FamilyID=BA73B169-A648-49AF-BC5E-A2EEBB74C16B* (sorry, aber die Adressen hat Microsoft so gewählt ...) downloaden können. Sie finden den Link zu dem Dokumenten-Download auch in der Datei *Links.html* auf der Buch-CD.

Einschränkungen

UAC führt zu einigen Einschränkungen für Anwendungen:

- Ohne Anhebung des Sicherheits-Levels (die, wie unten beschrieben, nicht immer ausgeführt wird) kann eine Anwendung keine administrativen Aufgaben ausführen. Außerdem sind die Rechte für bestimmte System-Ordner meist auf das Lesen eingeschränkt.
- Wird das Sicherheits-Level einer Anwendung von UAC angehoben, führt dies in der Regel zu Problemen bei der Kommunikation zwischen Fenstern, z. B. beim Drag&Drop. UAC erlaubt nämlich nicht, dass ein Fenster Nachrichten an ein Fenster mit einer höheren Priorität sendet. Als eines der wichtigsten Probleme dieses Sicherheits-Features sehe ich, dass unter UAC Drag&Drop zwischen einem Explorer-Fenster und einer höher gestuften Anwendung nicht mehr funktioniert.

Anhebung des Sicherheits-Levels

Das Anheben des Sicherheits-Level bei der Ausführung von Aktionen, die administrative Rechte erfordern, funktioniert bei einigen neueren Microsoft-Anwendungen relativ problemlos. Sie können das mit dem Windows-Explorer ausprobieren, indem Sie versuchen, im Windows-Ordner eine Datei oder einen Ordner anzulegen. Da ein Standardbenutzer (normalerweise) keine Schreibrechte im Windows-Ordner besitzt, fragt UAC ab, ob die Aktion mit den Rechten eines Administrators ausgeführt werden soll.

UAC überprüft auch, ob eine Anwendung eine Installations- oder Update-Anwendung ist, was an verschiedenen Indikatoren wie z. B. dem Text »install«, »setup« oder »update« im Dateinamen der Anwendung erkannt wird. Für diese Art von Anwendungen erfolgt dann auch eine Anhebung des Sicherheits-Levels.

Hinweis

Für andere Anwendungen, und dazu gehört nach meinen Tests auch z. B. Word 2007, erfolgt allerdings keine automatische Anhebung. Dies gilt zunächst einmal auch für normale .NET-Anwendungen: Für diese wird beim Versuch eine administrative Aufgabe auszuführen oder in einen Ordner zu schreiben, der für Standardbenutzer schreibgeschützt ist, eine `SecurityException` generiert.

Wenn Sie eine .NET-Anwendung entwickeln, die administrative Aufgaben ausführen soll, müssen Sie die gewünschte Anhebung des Sicherheits-Levels im Manifest der Anwendung deklarieren.

In Visual Studio fügen Sie dem Projekt dazu eine Anwendungs-Manifest-Datei (Application manifest file) hinzu. Wenn Sie die Rechte der Anwendung auf Administrator-Rechte anheben lassen wollen, sieht der Inhalt der Manifestdatei folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0"
  xmlns="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <requestedExecutionLevel level="requireAdministrator" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</asmv1:assembly>
```

Listing 134: Inhalt einer Manifestdatei zur Anhebung des Sicherheits-Levels der Anwendung auf das Niveau eines Administrators

Im `requestedExecutionLevel`-Element stellen Sie über das `level`-Attribut den angeforderten Sicherheits-Level ein. Möglich sind hier die folgenden Einstellungen:

- `asInvoker`: Die Anwendung wird grundsätzlich unter dem Sicherheits-Token des Prozesses ausgeführt, unter dem die Anwendung gestartet wurde.
- `highestAvailable`: Die Anwendung fordert an, dass sie unter dem höchsten Sicherheits-Token ausgeführt wird, das der aktuelle Benutzer anfordern kann.
- `requireAdministrator`: Die Anwendung fordert an, dass sie unter einem Administrator-Token ausgeführt wird.

Hinweis

Beachten Sie, dass das Anheben der Sicherheits-Levels beim Testen unter Visual Studio nicht funktioniert, da die Anwendung in diesem Fall unter dem Visual-Studio-Host-Prozess und Visual Studio nur unter dem Sicherheits-Token eines Standardbenutzers ausgeführt wird. Wenn Sie das Anheben testen wollen, müssen Sie das entsprechende Projekt unter Visual Studio mit STRG + F5 starten (was dazu führt, dass das Sicherheits-Level von Visual Studio angehoben wird) oder die erzeugte .exe-Datei direkt ausführen.

Hinweis

Beachten Sie, dass Sie die Anhebung des Sicherheits-Levels nur in Sonderfällen anfordern und in der Regel Anwendungen entwickeln sollten, die keine höheren Rechte als die eines Standardbenutzers erfordern.

Entwickeln von Anwendungen unter Rücksichtnahme auf UAC

Microsoft empfiehlt, Anwendungen grundsätzlich so zu entwickeln, dass diese keine höheren Rechte als die eines Standardbenutzers erfordern. Wenn Sie dies berücksichtigen, spielt UAC für Ihre Anwendungen keine Rolle.

Ihre Anwendungen sollten daher die folgenden Regeln einhalten:

- Sie sollten keine administrativen Aufgaben ausführen, wie zum Beispiel beim Start den Setup-Prozess vervollständigen.
- Sie sollten nicht direkt in das Windows-Verzeichnis, das Programm-Verzeichnis oder in deren Unterverzeichnisse schreiben.
- Die Anwendungen sollten Konfigurations- oder andere Daten, die geschrieben werden müssen, grundsätzlich in dem privaten Ordner des aktuellen Benutzers (unter Vista: C:\Users\<Benutzername>) oder im Daten-Ordner für alle Benutzer (unter Vista: C:\Users\All Users) verwalten.
- Ein automatisches (Online-)Update sollte eine Technik verwenden, die für Standardbenutzer verfügbar ist, wie z. B. das *Windows Installer 4.0 User Account Control Patching.*
- Die Anwendung sollte keine hart-codierten Pfade verwenden.

Hinweis

Zum Testen der Anforderung einer Anwendung auf administrative Anforderungen können Sie den von Microsoft bereitgestellten *Microsoft Standard User Analyzer* verwenden, den Sie im *Microsoft Application Compatibility Toolkit 5.0* finden. Laden Sie dieses an der Adresse *www.microsoft.com/downloads/details.aspx?FamilyID=24da89e9-b581-47b0-b45e-492dd6da2971* herunter (den Link finden Sie wie immer auch in der Datei *Links.html* auf der Buch-CD).

84 Ausnahmen global behandeln

Ausnahmen, die in der Anwendung nicht behandelt werden, werden von der CLR in einem einfachen Dialog gemeldet. Dieser Dialog (der bei `Windows.Forms`-Anwendungen per Voreinstellung ein anderer ist als bei Konsolen- und WPF-Anwendungen) ist mehr oder weniger aussagekräftig, wie Abbildung 55 und Abbildung 56 zeigen.



Abbildung 55: Meldung der CLR bei einer unbehandelten Ausnahme in einer Windows.Forms-Anwendung ohne Debug-Modus



Abbildung 56: Meldung der CLR bei einer unbehandelten Ausnahme in einer WPF- oder Konsolenanwendung oder in einer Windows.Forms-Anwendung, die zum Debugging eingestellt ist

Der DEBUGGEN-Schalter im neueren Dialog (Abbildung 56) ist nur dann vorhanden, wenn auf dem System ein Debugger installiert ist. Dies ist nur dann der Fall, wenn das .NET-Framework-SDK installiert ist. Über diesen Schalter kann die Anwendung debugt werden. Der Debugger berücksichtigt aber für den Fall, dass der Quellcode nicht im Ordner der Anwendung gespeichert ist (was ja in der Regel nicht der Fall ist), nur den CIL-Code. Damit können nur absolute System-Profis etwas anfangen.

Hinweis

Für `Windows.Forms`-Anwendungen können Sie in der Konfiguration einstellen, dass statt des Standard-Dialogs (Abbildung 55) der neuere Dialog (Abbildung 56) mit der Möglichkeit, zu debuggen, angezeigt wird. Dazu setzen Sie in der Maschinen- oder Anwendungskonfiguration im Element *system.windows.forms* das Attribut *jitDebugging* auf `true`. Zusätzlich dazu muss die Anwendung im Debug-Modus kompiliert worden sein.

Die meisten Benutzer werden wohl mit diesen Möglichkeiten überfordert sein. Außerdem kann es sein, dass beim Beenden der Anwendung Daten verloren gehen. Um dies zu verhindern können Sie selbstverständlich alle Ausnahmen im Programm explizit behandeln, was Sie dadurch erreichen, dass Sie zumindest in jeder Ereignismethode eine Ausnahmebehandlung implementieren. Eigentlich ist dies auch der bessere Weg, da Sie dem Benutzer dann genauere Informationen über den Kontext des Fehlers geben können.

Alternativ können Sie unbehandelte Ausnahmen aber auch global abfangen. Basis dieser Technik ist für WPF-Anwendungen das Ereignis `DispatcherUnhandledException` des Objekts, das die Anwendung repräsentiert. In einer `Windows.Forms`-Anwendung verwenden Sie das statische Ereignis `OnThreadException` der Klasse `Windows.Forms.Application`. Diese Ereignisse werden für alle unbehandelten Ausnahmen aufgerufen, die im UI-Thread der Anwendung eintreten.

Achtung

Diese Ereignisse werden allerdings nicht für unbehandelte Ausnahmen aufgerufen, die in einem anderen als dem UI-Thread auftreten. In eigenen Arbeits-Threads sollten Sie also trotz globaler Ausnahmebehandlung immer eine eigene Ausnahmebehandlung vorsehen.

In einer WPF-Anwendung erreichen Sie das `DispatcherUnhandledException`-Ereignis über die von `Application` abgeleitete Klasse (normalerweise die `App`-Klasse), die die WPF-Anwendung repräsentiert. Im Konstruktor können Sie das Ereignis zuweisen. Die Eigenschaft `Exception` des Ereignisargument-Objekts liefert die aufgetretene Ausnahme. Nach der Verarbeitung der Ausnahme müssen Sie die Eigenschaft `Handled` des übergebenen Ereignisargument-Objekts auf `true` setzen, damit die Ausnahme nicht an die CLR weitergegeben wird:

```
public partial class App : Application
{
   /* Konstruktor */
   public App()
   {
```

Listing 135: Globale Ausnahmebehandlung in einer WPF-Anwendung

```
      // Zuweisen der globalen Ausnahmebehandlung für den UI-Thread
      this.DispatcherUnhandledException +=  new System.Windows.Threading.
         DispatcherUnhandledExceptionEventHandler(
         this.App_DispatcherUnhandledException);
   }

   /* Behandelt alle unbehandelten Ausnahmen, die im UI-Thread auftreten.
      Behandelt keine unbehandelten Ausnahmen, die in anderen Threads
      auftreten! */
   private void App_DispatcherUnhandledException(object sender,
      System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)
   {
      // Ausnahme ausgeben
      MessageBox.Show("Unerwarteter Fehler: " + e.Exception.Message,
         "Globale Ausnahmebehandlung in WPF", MessageBoxButton.OK,
         MessageBoxImage.Error);

      // Das Ereignis als 'Behandelt' kennzeichnen, damit die Ausnahme
      // nicht an die CLR weitergegeben wird
      e.Handled = true;

      // Hier können (und sollten) Sie Ausnahmen protokollieren.
      // Wenn Sie den Stack-Trace protokollieren, hilft diese enorm
      // bei der späteren Fehlersuche.
   }
}
```

Listing 135: Globale Ausnahmebehandlung in einer WPF-Anwendung (Forts.)

In einer `Windows.Forms`-Anwendung weisen Sie dem `OnThreadException`-Ereignis in der `Main`-Methode (in *Program.cs*) eine passende Ereignisbehandlungsmethode zu. In `OnThreadException` existiert keine Handled-Eigenschaft, die Sie auf `true` setzen müssen (wie in WPF). Wenn in einer `Windows.Forms`-Anwendung Ausnahmen global abgefangen werden, werden diese implizit nicht an die CLR weitergegeben.

```
static class Program
{
   [STAThread]
   static void Main()
   {
      // Zuweisen der globalen Ausnahmebehandlung für den UI-Thread
      Application.ThreadException +=
         new System.Threading.ThreadExceptionEventHandler(
         Application_ThreadException);

      Application.EnableVisualStyles();
```

Listing 136: Globale Ausnahmebehandlung in einer Windows.Forms-Anwendung

```
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new MainForm());
   }

   /* Behandelt alle unbehandelten Ausnahmen, die im UI-Thread auftreten.
      Behandelt keine unbehandelten Ausnahmen, die in anderen Threads
      auftreten! */
   private static void Application_ThreadException(object sender,
      System.Threading.ThreadExceptionEventArgs e)
   {
      // Ausnahme ausgeben
      MessageBox.Show("Unerwarteter Fehler: " + e.Exception.Message,
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);

      // Hier können (und sollten) Sie Ausnahmen protokollieren.
      // Wenn Sie den Stack-Trace protokollieren, hilft dies enorm
      // bei der späteren Fehlersuche.
   }
}
```

Listing 136: Globale Ausnahmebehandlung in einer Windows.Forms-Anwendung (Forts.)

Tipp

Falls Sie im Programm alle erwarteten Ausnahmen explizit abfragen und in der Methode für die globale Ereignisbehandlung eigentlich nur unerwartete Ausnahmen abgefangen werden, sollten Sie dem Anwender neben einer Information über den aufgetretenen Fehler die Möglichkeit geben, die Details (der Ausnahme-Meldung inkl. aller inneren Ausnahmen und den Stack-Trace) an eine Support-E-Mail-Adresse zu mailen (siehe Rezept 211). Um genauere Informationen über die Ausnahme zu erhalten sollten Sie dem Release der Anwendung die automatisch erstellte Debug-Informationsdatei (mit der Endung *.pdb*) mitliefern. So erhalten Sie im Stack-Trace zusätzliche Informationen über die Quelle des Fehlers (inklusive der Zeilennummer). Diese Informationen erleichtern das Debuggen von Fehlern, die lediglich beim Anwender auftreten.

85 Den Speicherverbrauch von .NET-Anwendungen verstehen und gegebenenfalls reduzieren

Schaut man sich die Speicherauslastung von .NET-Anwendungen im Windows-Taskmanager an, so scheint es, als würden diese Anwendungen relativ viel Speicher verbrauchen. Bereits eine minimale »Hello World«-Anwendung belegt in der Regel mehrere Megabyte. Größere Anwendungen kommen schnell auf ein Vielfaches von zehn oder sogar ein Vielfaches von einhundert Megabyte (besonders wenn diese fahrlässig programmiert wurden).

Die Speicherauslastung, die im Taskmanager angezeigt wird, zeigt jedoch nicht den tatsächlichen Speicherverbrauch der Anwendung. Vielmehr handelt es sich dabei um den für die Anwendung reservierten Speicher. Die CLR reserviert für eine laufende Anwendung relativ großzügig Speicher, damit bei der Ausführung der Anwendung auf das inperformante Neu-

Reservieren und Umschichten des Speichers verzichtet werden kann. Dabei wird natürlich der im Moment verfügbare Arbeitsspeicher berücksichtigt. Ist dieser relativ klein, erhält die Anwendung nur relativ wenig Speicher. Hinzu kommt, dass auch Windows aus Performancegründen relativ großzügig mit der Reservierung von Speicher umgeht. Im Prinzip brauchen Sie sich also um den Arbeitsspeicher keine Gedanken zu machen.

Wichtig ist allerdings, dass Sie in Ihren Programmen nach der Verwendung von Objekten deren `Dispose`-Methode aufrufen, sofern diese eine solche besitzen. `Dispose` sorgt dafür, dass externe Ressourcen freigegeben werden. Diese Methode wird zwar auch im Destruktor der Objekte, die eine solche besitzen, aufgerufen (sofern die entsprechenden Klassen sorgfältig entwickelt wurden). Die CLR stellt aber nicht unbedingt sicher, dass der Destruktor (der ja eigentlich nur eine Finalisierungsmethode ist) aufgerufen wird. Das gilt im Besonderen für kleine Objekte wie die, die beim Zeichnen mit GDI+ verwendet werden (siehe Rezept 338).

Um sich die Arbeit zu vereinfachen können Sie Objekte, die eine `Dispose`-Methode besitzen, auch in einer `using`-Direktive erzeugen:

```
string sourceFilename = Path.Combine(
   Application.StartupPath, "Les Crosets.jpg");
using (Bitmap sourceBitmap = new Bitmap(sourceFilename))
{
   // Bitmap bearbeiten
}
```

Listing 137: Verwenden der using-Direktive

Der Compiler setzt die `using`-Direktive so um, dass automatisch hinter allen Anweisungen, die in den Block-Klammern angegeben werden, auf eine sichere Weise `Dispose` aufgerufen wird. Der resultierende IL-Code würde dann zurückübersetzt in etwa folgendermaßen aussehen:

```
string sourceFilename = Path.Combine(
   Application.StartupPath, "Les Crosets.jpg");
Bitmap sourceBitmap = null;
try
{
   sourceBitmap = new Bitmap(sourceFilename))

   // Bitmap bearbeiten
}
finally
{
   if (sourceBitmap != null)
   {
      try
      {
         sourceBitmap.Dispose();
```

Listing 138: Das, was der Compiler aus einem using-Block macht

```
      }
      catch {}
   }
}
```

Listing 138: Das, was der Compiler aus einem using-Block macht (Forts.)

Über die sehr einfach anzuwendende `using`-Direktive stellen Sie sicher, dass Ihre Programme die teilweise sehr umfangreichen externen Ressourcen, die manche Objekte verwenden, explizit freigeben und damit möglichst wenig Speicher verbrauchen.

Falls Ihr Programm trotz des korrekten Aufrufs von `Dispose` zu viel Speicher »verbraucht«, können Sie zunächst versuchen die `Collect`-Methode des Garbage Collectors aufzurufen, den Sie über `System.GC` erreichen. `GC.Collect` »erzwingt« eine Garbage Collection. Diese Methode sorgt dafür, dass der Garbage Collector aufgefordert wird, seine Arbeit auszuführen und die Objekte, die nicht mehr referenziert werden, freizugeben. Dies führt jedoch nicht unbedingt dazu, dass die nicht mehr referenzierten Objekte auch wirklich freigegeben werden, denn der Garbage Collector führt seine Arbeit nur dann aus, wenn sich eine Garbage Collection lohnt und sich das Programm gerade in einem untätigen Zustand befindet.

Wenn Sie nichts übergeben, sorgt `Collect` für eine Müllsammlung in allen Generationen des GC. Sie können aber auch den Index der Generation übergeben, die behandelt werden soll. Eine Müllsammlung in der Generation 0 wird wesentlich schneller ausgeführt als eine Müllsammlung in allen Generationen:

```
GC.Collect(0);
```

In der Praxis würde ich aber immer alle Generationen behandeln, da der Sinn eines manuellen Aufrufs wohl sein sollte, möglichst viel Speicher freizugeben.

Wenn Sie `Collect` aufrufen, sollten Sie danach über die Methode `WaitForPendingFinalizers` dafür sorgen, dass der Finialisierer-Aufruf-Thread seine Arbeit ausgeführt hat. Danach sollten Sie `Collect` noch einmal aufrufen:

```
GC.Collect();
GC.WaitForPendingFinalizers();
GC.Collect();
```

In absolut hartnäckigen Fällen können Sie auch die API-Funktion `SetProcessWorkingSetSize` aufrufen. Diese Funktion, die nur in Windows NT und ab Windows XP zur Verfügung steht, trimmt das von einer Anwendung reservierte *Working Set* (Set von physikalischen Speicherseiten). Am ersten Argument übergeben Sie den Handle des Prozesses Ihrer Anwendung, am zweiten Argument einen Byte-Minimalwert für die verbleibenden Speicherseiten und am dritten einen Maximalwert. Wenn Sie für den Minimal- und den Maximalwert -1 übergeben, reduziert `SetProcessWorkingSetSize` den reservierten physikalischen Arbeitsspeicher für den angegebenen Prozess kurzzeitig auf 0. Die CLR und Windows sorgen aber natürlich dafür, dass der reservierte Speicher sofort wieder auf einen für den aktuellen Zustand ausreichenden Betrag erhöht wird. Damit erreichen Sie, dass der reservierte Arbeitsspeicher minimiert wird. Im Prinzip entspricht das Ergebnis eines Aufrufs von `SetProcessWorkingSetSize` mit -1 an den beiden letzten Argumenten dem Minimieren und dem nachfolgenden Wiederherstellen des Hauptfensters einer Anwendung. Wenn Sie dies einmal ausprobieren, während Sie den Spei-

cherverbrauch der Anwendung im Taskmanager beobachten, werden Sie feststellen, dass dieser nach dem Minimieren und Wiederherstellen in der Regel enorm verkleinert wird.

`SetProcessWorkingSetSize` wird folgendermaßen deklariert:

```
[System.Runtime.InteropServices.DllImport("kernel32.dll")]
private static extern bool SetProcessWorkingSetSize(
   IntPtr procHandle, int min, int max);
```

Listing 139 zeigt eine Methode, die den reservierten Arbeitsspeicher für die aktuelle Anwendung auf ein Minimum reduziert. Diese Methode, die den Import der Namensräume `System` und `System.Diagnostics` erfordert, versucht zunächst die nicht mehr referenzierten Objekte freizugeben. Erst danach wird die `SetProcessWorkingSetSize`-Funktion aufgerufen um den reservierten Arbeitsspeicher zu reduzieren. Da diese Funktion nur in der NT-Linie der Microsoft-Betriebssysteme zur Verfügung steht, überprüft `ReduceMemoryUsage`, ob die aktuelle Plattform die Windows-32-Bit-NT-Plattform ist.

```
public static void ReduceMemoryUsage()
{
   // Garbage Collection erzwingen um freie Objekte zu zerstören
   GC.Collect();

   // Auf noch arbeitende Finalisierer warten
   GC.WaitForPendingFinalizers();

   // Den reservierten Arbeitsspeicher minimieren
   if (Environment.OSVersion.Platform == PlatformID.Win32NT)
   {
      SetProcessWorkingSetSize(
         Process.GetCurrentProcess().Handle, -1, -1);
   }
}
```

Listing 139: Minimieren des von der aktuellen Anwendung reservierten Arbeitsspeichers

Achtung

Sie sollten vorsichtig mit `ReduceMemoryUsage` umgehen. Zunächst kostet der Aufruf der `Collect`-Methode des Garbage Collectors unter Umständen einiges an Zeit. Auch das nachfolgende Minimieren des reservierten Speichers ist natürlich zeitaufwändig. Außerdem sollten Sie diese Funktion immer nur dann aufrufen, wenn Ihre Anwendung gerade nicht weiter beschäftigt ist. Ansonsten könnte der Aufruf zu unerwarteten Nebeneffekten führen. Achten Sie also darauf, dass nicht gerade ein Arbeits-Thread parallel läuft, während Sie `ReduceMemoryUsage` aufrufen. Gegebenenfalls rufen Sie diese Methode nicht automatisch auf, sondern überlassen dies dem Anwender, zum Beispiel über einen Menüpunkt. Wenn Ihre Anwendung eigene Threads einsetzt, sollten Sie diese in einer Auflistung verwalten, vor dem Aufruf von `ReduceMemoryUsage` durchgehen und über die `IsAlive`-Eigenschaft überprüfen, ob einer Ihrer Threads gerade noch ausgeführt wird. In diesem Fall sollten Sie `ReduceMemoryUsage` nicht aufrufen. In dem Beispiel zu diesem Rezept finden Sie eine entsprechende Programmierung.

86 Konfigurationsdaten in der .config-Datei verwalten

Wie Sie sicher wissen, kann eine Anwendung in einer XML-Datei, deren Name dem Dateinamen der Anwendung mit angehängter Endung *.config* entspricht und die im Ordner der Anwendung gespeichert ist, Konfigurationsdaten verwalten.

Diese Datei ist grundlegend folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

</configuration>
```

Listing 140: Eine Minimal-Konfigurationsdatei

Im `configuration`-Element können verschiedene weitere Konfigurations-Elemente angelegt werden. Dabei können Sie die in der Datei *machine.config* vordefinierte Konfiguration für die Anwendung (in der Regel teilweise) überschreiben bzw. neu definieren. Dies soll hier allerdings kein Thema sein.

Zusätzlich dazu können Sie in der *.config*-Datei Ihre eigenen Konfigurationsdaten verwalten. Visual Studio 2008 macht diese Aufgabe leicht.

In Visual Studio 2008 wählen Sie dazu in den Eigenschaften des Projekts das Register EINSTELLUNGEN und stellen dort die Konfigurationswerte ein, die Sie benötigen. Dabei können Sie neben dem Namen auch den Datentyp einstellen und festlegen, welchen Geltungsbereich (Scope) die Einstellung besitzt.

Der Geltungsbereich »Application« bedeutet, dass diese Einstellung ausschließlich in der *.config*-Datei verwaltet wird, die im Anwendungsverzeichnis gespeichert ist. Solche Einstellungen können nur gelesen (also nicht in der Anwendung mit neuen Werten beschrieben) werden. Der Geltungsbereich »User« bewirkt, dass der Wert der Einstellung im Programm auch geändert und zurückgeschrieben werden kann. Das Zurückschreiben erfolgt per Voreinstellung allerdings nicht in die *.config*-Datei im Anwendungsverzeichnis, sondern in eine spezielle Konfigurationsdatei im Verzeichnis für die Anwendungsdaten des aktuellen Benutzers[1]. Lediglich die Default-Werte der Einstellung, die gelesen werden, wenn noch keine benutzerspezifische Konfigurationsdatei vorhanden ist, können in der *.config*-Datei im Anwendungsverzeichnis verwaltet werden.

1. Die Regeln zum Erhalt des »Designed for Windows XP«-Logo für eine Anwendung besagen u. a., dass Anwendungen Konfigurationsdaten nicht im Anwendungsverzeichnis speichern dürfen. Solche Daten müssen im Anwendungdaten-Verzeichnis des lokalen aktuellen Benutzers (normalerweise ist das das Verzeichnis *C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\Anwendungsdaten*), im äquivalenten Verzeichnis für »Roaming«-Benutzer (»wandernde« Benutzer) oder im Verzeichnis für alle Benutzer gespeichert werden.

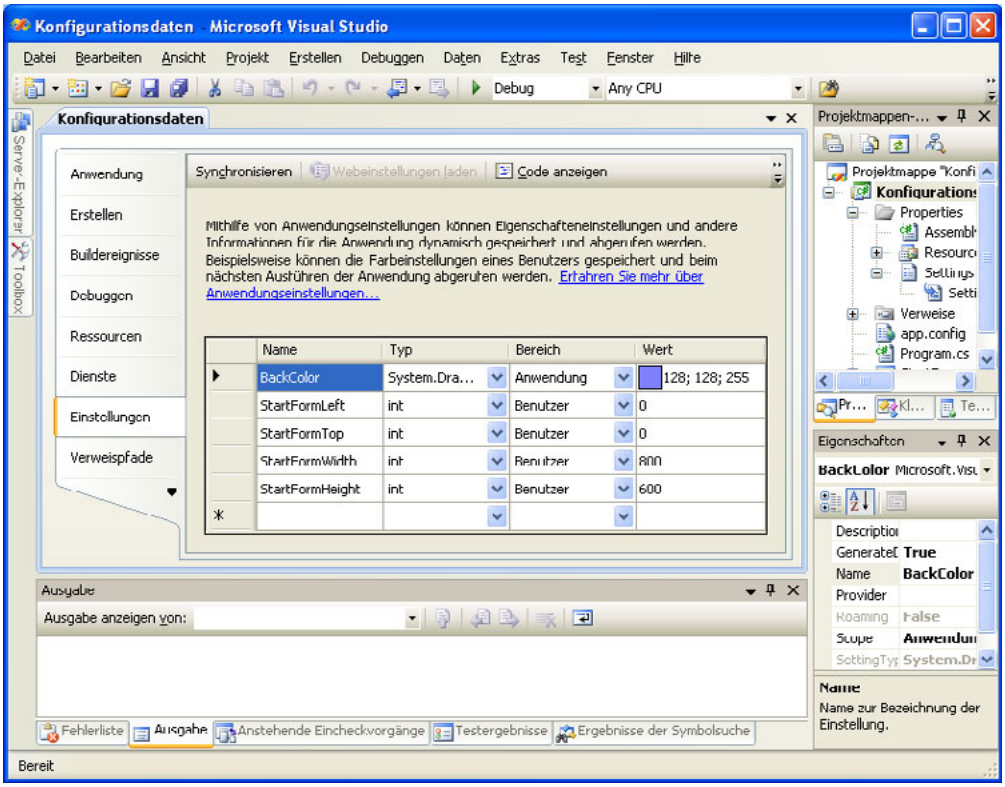


Abbildung 57: Bearbeiten von Konfigurationswerten in Visual Studio

Wenn Sie die geänderte Konfiguration speichern, passt Visual Studio die Konfigurationsdatei an und erzeugt eine Klasse `Settings` im Ordner *Properties*. Über eine Instanz dieser Klasse können Sie die Konfigurationsdaten typsicher lesen. Die Default-Instanz erreichen Sie über die `Default`-Eigenschaft der `Settings`-Klasse:

```
// Hintergrundfarbe des Formulars einlesen
this.BackColor = Properties.Settings.Default.BackColor;

// Position und Größe des Formulars auslesen
this.Left = Properties.Settings.Default.StartFormLeft;
this.Top = Properties.Settings.Default.StartFormTop;
this.Width = Properties.Settings.Default.StartFormWidth;
this.Height = Properties.Settings.Default.StartFormHeight;
```

Listing 141: Lesen von Konfigurationsdaten, die über eine Settings-Klasse verwaltet werden

Diejenigen Einstellungen, die den Geltungsbereich »User« besitzen, können Sie mit neuen Werten versehen und die geänderte Konfiguration über die `Save`-Methode speichern:

```
// Position und Größe des Formulars in der Konfiguration ablegen
Properties.Settings.Default.StartFormLeft = this.Left;
Properties.Settings.Default.StartFormTop = this.Top;
Properties.Settings.Default.StartFormWidth = this.Width;
Properties.Settings.Default.StartFormHeight = this.Height;

// Konfiguration speichern
try
{
   Properties.Settings.Default.Save();
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Speichern der Konfiguration: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Listing 142: Speichern von Konfigurationsdaten, die über eine Settings-Klasse verwaltet werden

Hinweis

Die `Settings`-Klasse verwendet Provider für das Lesen und Speichern der Daten. Per Default wird eine Instanz der Klasse `LocalFileSettingsProvider` eingesetzt. Diese verwaltet Konfigurationsdaten mit dem Geltungsbereich »User« im Ordner für lokale Benutzer (normalerweise ist das unter XP der Ordner *C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\Anwendungsdaten* und unter Vista der Ordner *C:\Users\<Benutzername>\AppData\Local*). In diesem Ordner wird mit einer gewöhnungsbedürftigen Logik ein Unterordner erzeugt. Dabei wird zunächst ein Ordner angelegt, der dem Schema *Firmenname\Dateiname der Assembly* entspricht. Ist das Assembly-Attribut `AssemblyCompany` nicht (in der Datei *Properties\AssemblyInfo.cs*) angegeben, entfällt der Ordner *Firmenname*. In diesem Ordner wird ein Ordner angelegt, dessen Name etwas eigenartig erscheint. Er beginnt mit den Dateinamen der Assembly (wobei die Endung nur mit zwei Stellen ausgegeben wird). Dann folgt »_Url_« und danach ein verschlüsselt angegebener URL. In diesem Ordner ist schließlich ein Ordner angelegt, der die Versionsnummer der Anwendung trägt, und in diesem eine Datei *user.config* gespeichert, die die Konfigurationsdaten mit dem Geltungsbereich »User« verwaltet. Etwas verwirrend, aber so können Konfigurationsdaten eindeutig einer bestimmten Anwendung bzw. einer bestimmten Version einer Anwendung zugeordnet werden. Der Debugger von Visual Studio 2008 ruft übrigens beim Start einer Windowsanwendung eine spezielle *.exe*-Datei auf, die den Namen der Anwendungsassembly mit *.vshost.exe* am Ende trägt. Die Konfigurationsdaten für eine so gestartete Anwendung werden in einem separaten Ordner verwaltet, in dessen Namen *.vshos_Url_* statt *.exe_Url_* angegeben ist. So ist sichergestellt, dass Sie die Konfiguration einer bereits produktiv laufenden Anwendung beim Testen nicht überschreiben.

Achtung

Die Einstellungen werden also zum einen Assembly-spezifisch und zum anderen ortsspezifisch gespeichert. Für eine normale Anwendung (keine ClickOnce-Anwendung), bedeutet dies:

- dass die Einstellungen verloren gehen, wenn die Anwendung in einen anderen Ordner kopiert wird,
- dass die Einstellungen verloren gehen, wenn die Anwendungsdatei umbenannt wird oder in den Assembly-Attributen der Firmenname geändert wird
- und dass die Einstellungen bei jeder Aktualisierung der Anwendung (mit einer neuen Versionsnummer) verloren gehen.

Besonders der letzte Punkt ist sehr ärgerlich (aber es gibt eine Lösung dafür). Sie können von einem Anwender nicht erwarten, dass er bei jeder Aktualisierung der Anwendung alle persönlichen Einstellungen erneut vornimmt.

Das Aktualisierungs-Problem gilt aber nur für normale Anwendungen. Für eine Anwendung, die per ClickOnce installiert wird, gehen die Einstellungen bei einem automatischen Update nicht verloren. Der Grund dafür ist, dass bei der (automatischen) Aktualisierung einer ClickOnce-Anwendung implizit die Methode `Upgrade` der Default-`Settings`-Instanz aufgerufen wird. Diese Methode liest die Einstellungen der vorherigen Version der Anwendung, die am besten zu der aktuellen Version passt, und schreibt diese in den Ordner der aktuellen Version.

Das ist auch der (relativ unbekannte und schlecht dokumentierte) Trick um zu erreichen, dass nach einer Aktualisierung einer Anwendung die Einstellungen nicht verloren gehen: Rufen Sie die `Upgrade`-Methode der `Settings`-Instanz auf. Die Frage ist nur, wann Sie das machen sollten.

Tipp

Der Trick dazu ist, in den Einstellungen eine spezielle boolesche Einstellung mit dem Bereich »Benutzer«, vielleicht mit dem Namen *CallUpgrade*, zu verwalten. Die Voreinstellung setzen Sie auf `true`. Beim Start der Anwendung überprüfen Sie, ob *CallUpgrade* `true` ist, und rufen in diesem Fall `Upgrade` auf. *CallUpgrade* setzen Sie dann auf `false` und speichern die Einstellungen:

```
if (Settings.Default.CallUpgrade)
{
   Settings.Default.Upgrade();
   Settings.Default.CallUpgrade = false;
   Settings.Default.Save();
}
```

Listing 143: Der ultimative Trick zum Lesen der Benutzer-Einstellungen nach dem Aktualisieren einer (normalen) Anwendung

Hinweis

Etwas problematisch an dieser Speicherung von Konfigurationsdaten ist zudem, dass die Daten automatisch nur beim Start der Anwendung ausgelesen werden. Werden die Daten in der Konfigurationsdatei geändert, während die Anwendung läuft, sind die Änderungen zunächst nicht sichtbar, auch wenn die Anwendung die Daten erneut abfragt. Dies kann in Anwendungen, die permanent im Hintergrund laufen (z. B. Windows-Dienste) zu einem Problem werden. Sie können jedoch in den Fällen, in denen geänderte Einstellungen in der Anwendung sichtbar sein müssen, vor dem Lesen einfach die `Reload`-Methode der `Settings`-Instanz aufrufen. Diese Methode führt dazu, dass die Einstellungen neu eingelesen werden.

Hinweis

Sie können auch eigene Provider implementieren, indem Sie eine Klasse entwickeln, die von der Klasse `SettingsProvider` abgeleitet ist und die die `IApplicationSettingsProvider`-Schnittstelle implementiert. So könnten Sie z. B. einen Provider entwickeln, der die Daten in einer »richtigen« XML-Datei verwaltet. Der neue Provider wird dann über das `SettingsProvider`-Attribut in der Settings-(Designer-)Datei angegeben, was aber zumindest für den Visual-Studio-2008-Designer etwas unklar ist, denn dieser überschreibt die Einstellungen in dieser Datei gnadenlos bei allen Änderungen (dagegen hilft ein weiterer Klassen-Teil der partiellen Klasse, der die Klassen entsprechend attributiert). Leider hat Microsoft dieses wichtige Thema nur sehr spärlich dokumentiert und liefert keine Beispiele. Bisher schlugen meine Versuche, einen eigenen Provider zu implementieren, leider immer fehl. In Internet war zu dieser Zeit auch nichts darüber zu finden. Falls Sie einen funktionierenden eigenen Provider entwickeln oder finden, wäre ich über eine E-Mail dankbar und würde das Ganze dann im Erratum veröffentlichen.

87 Konfigurationsdaten aus dem appSettings-Element der .config-Datei lesen

Neben der Möglichkeit, eine `Settings`-Instanz zu verwenden (siehe Rezept 86), können Sie Konfigurationsdaten aus der Konfigurationsdatei der Anwendung auch auf die alte (.NET-1.1-) Art verwalten. Dies besitzt in meinen Augen immer noch zwei kleine Vorteile: Zum einen sieht die Konfigurationsdatei wesentlich aufgeräumter aus als bei der Verwendung der `Settings`-Klasse und kann deswegen einfacher von Benutzern oder Administratoren verändert werden. Zum anderen können Sie ohne Probleme Konfigurationsdaten an einer zentralen Stelle verwalten, unabhängig davon, welcher Benutzer die Anwendung verwendet.

Die zu lesenden Daten müssen im Element `appSettings` unterhalb des `configuration`-Elements gespeichert sein:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appSettings>
      <add key="server" value="Zaphod" />
      <add key="userId" value="Trillian" />
```

Listing 144: Eine Beispiel-Konfigurationsdatei

```
      <add key="password" value="42" />
    </appSettings>
</configuration>
```

Listing 144: Eine Beispiel-Konfigurationsdatei (Forts.)

Die `add`-Elemente im `appSettings`-Element speichern anwendungsspezifische Daten. Diese Daten können Sie über die `AppSettings`-Eigenschaft der Klasse `ConfigurationManager` aus dem Namensraum `System.Configuration` auslesen:

```
string server = ConfigurationManager.AppSettings["Server"];
string userId = ConfigurationManager.AppSettings["UserId"];
string password = ConfigurationManager.AppSettings["password"];
```

Listing 145: Auslesen von Konfigurationsdaten aus dem appSettings-Element der .config-Datei

Das Programm benötigt dazu eine Referenz auf die Assembly *System.Configuration.dll*.

Hinweis

Wie ich es bereits in Rezept 86 beschrieben habe, leidet auch diese Art der Speicherung von Konfigurationsdateien an dem Problem, dass die Daten automatisch und nur beim Start der Anwendung ausgelesen werden. Werden die Daten in der Konfigurationsdatei geändert, während die Anwendung läuft, sind die Änderungen nicht sichtbar, auch wenn die Anwendung die Daten erneut abfragt. Als Lösung dieses Problems können Sie eine eigene XML-Datei für die Konfiguration verwenden, wie ich es im Rezept 90 zeige.

88 Externe Konfigurationsdateien lesen und schreiben

In einigen Fällen ist es notwendig, Konfigurationsdateien anderer Anwendungen zu lesen oder zu schreiben. Das ist z. B. der Fall, wenn Sie ein Setup für eine Anwendung erstellen, in dem der Anwender grundlegende Einstellungen vornehmen soll (die er z. B. in einem Windows-Installer-Setup über eine benutzerdefinierte Aktion eingibt). Das Setup-Programm muss dann die Konfigurationsdatei der installierten Anwendung mit den vom Anwender eingegebenen Daten überschreiben.

Dazu verwenden Sie die `ConfigurationManager`-Klasse (aus dem Namensraum `System.Configuration`), die es ermöglicht, alle Arten von .NET-Konfigurationsdateien (u. a. auch die *machine.config*) zu bearbeiten. Leider ist die Anwendung der `ConfigurationManager` für das Schreiben anderer als der eigenen Konfigurationsdateien zum einen nicht besonders gut dokumentiert und besitzt zum anderen zwei Fallstricke.

Dieses Rezept zeigt aber, wie Sie die Anwendungskonfiguration einer anderen Anwendung überschreiben können. Es arbeitet mit der folgenden Konfiguration, die einen Verbindungsstring, die anwendungsspezifische Einstellung »InstallationDate« und den Defaultwert der benutzerdefinierten Einstellung »DataFolder« verwaltet:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
```

```
        <sectionGroup name="applicationSettings" ... >
            <section name="Demo.Properties.Settings" ... />
        </sectionGroup>
        <sectionGroup name="userSettings" ... >
            <section name="Demo.Properties.Settings" ... />
        </sectionGroup>
    </configSections>


   <connectionStrings>
      <add name="DataConnection" connectionString=""/>
   </connectionStrings>
    <applicationSettings>
        <Demo.Properties.Settings>
            <setting name="InstallationDate" serializeAs="String">
                <value/>
            </setting>
        </Demo.Properties.Settings>
    </applicationSettings>
    <userSettings>
        <Demo.Properties.Settings>
            <setting name="DataFolder" serializeAs="String">
                <value/>
            </setting>
        </Demo.Properties.Settings>
    </userSettings>
</configuration>
```

Hinweis

Das Rezept zeigt lediglich, wie Sie die Konfiguration ändern, die im Anwendungsordner der anderen Anwendung verwaltet wird. Das Ändern einer benutzerspezifischen Konfigurationsdatei ist wesentlich komplexer, weil diese in einem speziellen Unterordner im Ordner der Anwendungsdaten für wandernde Benutzer (Roaming user) verwaltet wird. Ich denke aber nicht, dass das Schreiben in eine benutzerspezifische Konfigurationsdatei einer anderen Anwendung notwendig sein sollte, da Sie die Defaulteinstellungen der benutzerspezifischen Werte in der Konfiguration der Anwendung ändern können (wie dies in diesem Rezept auch realisiert ist).

Um die Anwendungs-Konfigurationsdatei einer anderen Anwendung zu bearbeiten, rufen Sie die statische Methode `OpenExeConfiguration` auf, der Sie den Pfad zu der *.exe*-Datei der anderen Anwendung übergeben. Diese Methode gibt ein `Configuration`-Objekt zurückgibt, über das Sie Zugriff auf die Konfiguration erhalten. Dabei arbeiten Sie immer mit Sektionen. Die Sektion für die Verbindungsstrings erreichen Sie z. B. über die `ConnectionStrings`-Eigenschaft, die ein `ConnectionStringsSection`-Objekt referenziert. Über dessen Eigenschaft `ConnectionStrings` können Sie die Verbindungsstrings lesen und schreiben.

Achtung

Hier wartet der erste Fallstrick: Wenn Sie lediglich einen vorhandenen Verbindungsstring neu definieren, indem Sie dessen Einstellungen überschreiben, führt dies nicht unbedingt [2] dazu, dass Ihre Änderungen später auch in die Konfigurationsdatei geschrieben werden. Das Problem liegt darin, dass der Konfigurations-Manager beim Ändern einer vorhandenen Einstellung grundsätzlich nicht erkennt, dass diese geändert wurde [3]. Sie sollten immer ein neues `ConnectionStringSettings`-Objekt erzeugen und dieses der `ConnectionStrings`-Auflistung zuweisen. Im einfachsten Fall löschen Sie vorher alle vorhandenen und fügen das neue einfach an.

```
// Konfiguration für die Anwendungsdaten der anderen Anwendung öffnen
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string exeFileName = "C:\\Demo\\Demo.exe";
Configuration configuration = ConfigurationManager.OpenExeConfiguration(
   exeFileName);

// Verbindungs-Einstellungen lesen
Console.WriteLine("Alte Verbindungsstring-Einstellung:");
ConnectionStringsSection csSection = configuration.ConnectionStrings;
if (csSection.ConnectionStrings["DataConnection"] != null)
{
   Console.WriteLine("Verbindung: " +
      csSection.ConnectionStrings["DataConnection"].ConnectionString);
}
else
{
   Console.WriteLine("Die Verbindungsstring-Einstellung " +
      "'DataConnection' existiert nicht");
}

// Verbindungs-Einstellung neu definieren
configuration.ConnectionStrings.ConnectionStrings.Clear();
configuration.ConnectionStrings.ConnectionStrings.Add(
   new ConnectionStringSettings("DataConnection",
   "Server=.\\SQLEXPRESS;Database=Demo;User=sa;Password=Galaxy"));
```

Listing 146: Lesen und Schreiben von Verbindungsstring-Einstellungen der Konfigurationsdatei einer anderen Anwendung

2. Dies hängt davon ab, mit welchem *saveMode* Sie die `Save`-Methode aufrufen.
3. Das liegt daran, dass eine Einstellung ein komplexes Objekt ist. Wenn eine Eigenschaft dieses separaten Objekts geändert wird, erhält der Konfigurationsmanager, der lediglich eine Referenz auf das Einstellungsobjekt hält, darüber keine Information.

Um die anwendungsspezifischen Einstellungen zu schreiben, benötigen Sie zunächst eine Referenz auf die Sektions-Gruppe, die diese Einstellungen verwaltet. Über diese ermitteln Sie dann eine Referenz auf die Sektion:

```
// Die Sektion einlesen, die die anwendungsspezifische
// Konfiguration enthält
ConfigurationSectionGroup applicationSettings =
   configuration.SectionGroups["applicationSettings"];
ClientSettingsSection settingSection =
   applicationSettings.Sections["Demo.Properties.Settings"]
   as ClientSettingsSection;
```

Listing 147: Ermitteln der Sektion für die anwendungsspezifischen Einstellungen

Leider können Sie dann nicht direkt auf die einzelnen Einstellungen zugreifen, die Sie über die `Settings`-Eigenschaft erreichen, da diese keinen Indexer bietet, dem Sie den Namen der Einstellung übergeben können. Sie müssen die einzelnen Einstellungen durchgehen und nach derjenigen suchen, die Sie verändern wollen.

Achtung

Und hier ist der zweite Fallstrick enthalten (der aber mit dem ersten zusammenhängt). Den Wert der Einstellung erhalten Sie über die Eigenschaft `Value`. `Value` referenziert ein `SettingValueElement`-Objekt. Dessen Eigenschaft `ValueXml` referenziert ein `XmlNode`-Objekt, das schließlich den Wert der Einstellung verwaltet. Leider können Sie nicht einfach nur die Eigenschaft `InnerText` dieses Objekts beschreiben, um den Wert zu ändern. In diesem Fall bekommt der Konfigurations-Manager die Änderung nicht mit und schreibt den geänderten Wert nicht in die Konfiguration.

Um dieses Problem zu lösen, müssen Sie der `ValueXml`-Eigenschaft ein neues `XmlNode`-Objekt zuweisen, das den geänderten Wert enthält. Da `XmlNode` keinen Konstruktor und auch keine Methode zur Erzeugung zur Verfügung stellt, müssen Sie zudem ein Dummy-`XmlDocument`-Objekt erzeugen, über das Sie die `XmlNode`-Instanz erzeugen können.

```
// XmlDocument-Dummy-Dokument zur Erstellung der XmlNode-Instanzen erzeugen
XmlDocument dummyDoc = new XmlDocument();

// Alle SettingElement-Instanzen durchgehen, um die Einstellungen
// zu finden, die geändert werden sollen
foreach (SettingElement settingElement in settingSection.Settings)
{
   if (settingElement.Name == "InstallationDate")
   {
      // Die Einstellung ändern, indem ein neues XML-Element
      // geschrieben wird. Dies ist leider notwendig, da der
```

Listing 148: Schreiben von anwendungsspezifischen Einstellungen

```
      // Konfigurationsmanager ansonsten nicht erkennt, dass
      // diese Daten geändert wurden, und die Daten einfach
      // nicht in die Konfiguration schreibt.
      settingElement.Value.ValueXml = dummyDoc.CreateElement("value");
      settingElement.Value.ValueXml.InnerText = DateTime.Now.ToString("u");
   }
}
```

Listing 148: Schreiben von anwendungsspezifischen Einstellungen (Forts.)

Für die benutzerspezifischen Einstellungen (bei denen Sie die Defaultwerte ändern können) gehen Sie ähnlich vor:

```
// Die Sektion einlesen, die die benutzerspezifische Konfiguration enthält
ConfigurationSectionGroup userSettings =
   configuration.SectionGroups["userSettings"];
settingSection =
   userSettings.Sections["Demo.Properties.Settings"]
   as ClientSettingsSection;

// Alle SettingElement-Instanzen durchgehen, um die Einstellungen
// zu finden, die geändert werden sollen
foreach (SettingElement settingElement in settingSection.Settings)
{
   if (settingElement.Name == "DataFolder")
   {
      settingElement.Value.ValueXml = dummyDoc.CreateElement("value");
      settingElement.Value.ValueXml.InnerText = "C:\\Data";
   }
}
```

Listing 149: Schreiben der Defaultwerte von benutzerspezifischen Einstellungen

Schließlich speichern Sie Ihre Änderungen über die `Save`-Methode. Dieser Methode können Sie einen Wert der `ConfigurationSaveMode`-Aufzählung übergeben:

- `Minimal`: Bewirkt, dass nur Einstellungen geschrieben werden, die sich von den geerbten Werten unterscheiden
- `Full`: Bewirkt, dass alle Einstellungen (auch geerbte) in die Konfiguration geschrieben werden
- `Modified`: Bewirkt, dass alle geänderten Einstellungen in die Konfiguration geschrieben werden, auch wenn deren Wert mit dem Wert einer geerbten Einstellung identisch ist

Für unseren Fall ist `Modified` am besten geeignet. Dabei gehen eventuell vorhandene Einstellungen, die Sie nicht ändern, nicht verloren und die Konfigurationsdatei bleibt prinzipiell mit der vorhergehenden identisch. Da `Modified` die Voreinstellung ist (was allerdings nicht dokumentiert ist), können Sie `Save` auch ohne Argument aufrufen:

```
// Konfiguration speichern
configuration.Save();
```

Listing 150: Speichern der geänderten Konfiguration

89 Konfigurationsdaten in eigenen Abschnitten speichern

Die Konfigurationsdaten einer Anwendung können Sie zwar relativ einfach in den Anwendungs-Einstellungen (`Settings`) speichern, beim Entwickeln von Komponenten ist es aber häufig notwendig, diese Daten dem Anwender oder Programmierer über die normale Anwendungskonfigurationsdatei verfügbar zu machen. In diesem Fall macht es Sinn, die Konfigurationsdaten in einem eigenen Abschnitt der *.config*-Datei zu speichern, der wesentlich einfacher zu pflegen ist als das mittlerweile veraltete `appSettings`-Element.

Zur Verwaltung von Konfigurationsdaten in einer eigenen Sektion müssen Sie einen Handler entwickeln. Ein solcher implementiert die Schnittstelle `IConfigurationSectionHandler`, deren Methode `Create` für alle registrierten, benutzerdefinierten Konfigurationsabschnitte aufgerufen wird. Über das Argument `section`, das ein `XmlNode`-Objekt referenziert, erhalten Sie Zugriff auf das XML-Element, das die Daten verwaltet.

In der Anwendung müssen (eigentlich erst später) die neuen Abschnitte in der Konfigurationsdatei registriert werden. Das folgende Listing registriert die Abschnitte *system* und *database* im Element *codebook*:

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

   <!-- Bekanntmachung der eigenen Konfigurations-Sektionen -->
   <configSections>
      <sectionGroup name="codebook">
         <section
           name="database"
           type="Addison_Wesley.Codebook.Configuration.
           ConfigSectionHandler, Config-Handler"/>
         <section
           name="system"
           type="Addison_Wesley.Codebook.Configuration.
           ConfigSectionHandler, Config-Handler"/>
      </sectionGroup>
   </configSections>
```

Listing 151: Registration eigener Konfigurations-Abschnitte

Das `sectionGroup`-Element legt über das Attribut `name` den Namen des XML-Elements fest, das die Konfigurationsdaten verwaltet. Im Element `section` werden die einzelnen Abschnitte definiert. Das Attribut `name` bestimmt dabei den Namen des XML-Elements. Das Attribut `type` legt fest, welche Klasse für das Handling dieser Elemente zuständig ist. Dabei geben Sie zuerst den

vollen Namen der Klasse (inklusive Namensraum) und durch ein Komma getrennt die Assembly an, in der diese Klasse gespeichert ist (im Beispiel *Config-Handler*). Das `sectionGroup`-Element kann (natürlich) auch komplett in der Haupt-Konfigurationsdatei *machine.config* gespeichert werden, was u. U. Sinn macht, wenn es sich um Konfigurationsdaten handelt, die Sie öfter verwenden. Dann müssen Sie aber auch dafür sorgen, dass diese Eintragung in der Maschinenkonfiguration auf den Computern eingetragen wird, auf dem Ihr Programm ausgeführt werden soll.

In der Konfigurationsdatei folgen dann Ihre eigenen Konfigurationssektionen:

```
    <codebook>
      <database>
        <server>Zaphod</server>
        <userId>Trillian</userId>
        <password>42</password>
      </database>

      <system>
        <lastAccess>01.01.2003 10:00:00</lastAccess>
      </system>
    </codebook>

</configuration>
```

Listing 152: Eigene Konfigurations-Abschnitte

Zur Verwaltung der später eingelesenen Daten benötigen Sie zunächst für jeden benutzerdefinierten Konfigurationsabschnitt eine Klasse.

```
/* Klasse zur Speicherung der Datenbank-Konfiguration */
public class DatabaseSection
{
   public string Server;
   public string UserId;
   public string Password;

   internal DatabaseSection(string server, string userId, string password)
   {
      this.Server = server;
      this.UserId = userId;
      this.Password = password;
   }
}

/* Klasse zur Speicherung der System-Konfiguration */
public class SystemSection
```

Listing 153: Klassen zur Speicherung der eingelesenen Daten

```
{
   public string LastAccess;

   internal SystemSection(string lastAccess)
   {
      this.LastAccess = lastAccess;
   }
}
```

Listing 153: Klassen zur Speicherung der eingelesenen Daten (Forts.)

Zum Lesen der eigenen Konfigurationsabschnitte müssen Sie eine Klasse entwickeln, die die Schnittstelle `IConfigurationSectionHandler` implementiert. Der `Create`-Methode dieser Klasse werden später Informationen über das übergeordnete Element, ein Konfigurationskontext (nur für ASP.NET-Anwendungen, wo dieser den virtuellen Pfad der Konfigurationsdatei enthält) und ein `XmlNode`-Objekt übergeben, das das XML-Element mit den Konfigurationsdaten enthält. Über dieses Objekt können Sie die Daten lesen.

Die Methode `Create` in Listing 154 fragt zunächst über die `Name`-Eigenschaft ab, um welchen Abschnitt es sich handelt. Dann werden die entsprechenden Daten eingelesen und damit eine Instanz der Klasse erzeugt, die für diese Daten vorgesehen ist. Diese Instanz gibt `Create` dann zurück.

Etwas problematisch ist, dass Konfigurationsdaten auch fehlen können. `Create` fragt deshalb vor dem Lesen ab, ob die einzelnen Unterelemente vorhanden sind. Für den Fall, dass ein unbekannter Konfigurationsabschnitt registriert wurde, generiert die Methode eine Ausnahme:

```
/* Klasse, die den Konfigurations-Sektions-Handler implementiert */
public class ConfigSectionHandler: IConfigurationSectionHandler
{
   /* Implementierung der Create-Methode */
   public object Create(object parent, object configContext, XmlNode section)
   {
      if (section.Name == "database")
      {
         // Einlesen der Unterelemente der Sektion
         XmlNode subNode;
         string server = null;
         subNode = section.SelectSingleNode("server");
         if (subNode != null)
         {
            server = subNode.InnerText;
         }

         string userId = null;
         subNode = section.SelectSingleNode("userId");
```

Listing 154: Handler für eigene Konfigurationsabschnitte

```
            if (subNode != null)
            {
               userId = subNode.InnerText;
            }

            string password = null;
            subNode = section.SelectSingleNode("password");
            if (subNode != null)
            {
               password = subNode.InnerText;
            }

            // Neue DatabaseConfig-Instanz zurückgeben
            return new DatabaseSection(server, userId, password);

         }
         else if (section.Name == "system")
         {
            // Einlesen der Unterelemente der Sektion
            XmlNode subNode;
            string lastAccess = null;
            subNode = section.SelectSingleNode("lastAccess");
            if (subNode != null)
            {
               lastAccess = subNode.InnerText;
            }

            // Neue SystemConfig-Instanz zurückgeben
            return new SystemSection(lastAccess);
         }
         else
         {
            // Unbekannte Sektion: Ausnahme werfen
            throw new ConfigurationErrorsException("Unbekannte Konfigurations-" +
               "Sektion '" + section.Name + "'");
         }
      }
   }
```

Listing 154: Handler für eigene Konfigurationsabschnitte (Forts.)

Nun müssen Sie die Konfigurationsdaten in der von Ihnen entwickelten Komponente lediglich noch über die Methode `ConfigurationSettings.GetConfig` einlesen, indem Sie den relativen Pfad zum Konfigurationsabschnitt angeben. `GetConfig` gibt das in `Create` erzeugte Objekt zurück, wenn der Abschnitt gefunden wird. Im anderen Fall wird `null` zurückgegeben. So können Sie beim Einlesen überprüfen, ob der Abschnitt vorhanden ist. Das Beispiel-Programm in

Listing 155 implementiert dies in Form einer einfachen Konsolenanwendung. Sie müssen die Namensräume `System` und `System.Configuration` importieren, um dieses Programm kompilieren zu können.

```
DatabaseSection databaseSection = null;
SystemSection systemSection = null;

try
{
   // Einlesen der Konfiguration
   systemSection = (SystemSection)ConfigurationManager.GetSection(
      "codebook/system");
   databaseSection = (DatabaseSection)ConfigurationManager.GetSection(
      "codebook/database");

   // Überprüfen, ob der System-Konfigurationsabschnitt eingelesen wurde
   if (systemSection != null)
   {
      // Ausgeben der Konfigurationsdaten
      Console.WriteLine("LastAccess: {0}", systemSection.LastAccess);
   }
   else
   {
      Console.WriteLine("Die Systemkonfiguration konnte nicht " +
         "eingelesen werden");
   }

   // Überprüfen, ob der Datenbank-Konfigurationsabschnitt eingelesen wurde
   if (databaseSection != null)
   {
      // Ausgeben der Konfigurationsdaten
      Console.WriteLine("Server: {0}", databaseSection.Server);
      Console.WriteLine("UserId: {0}", databaseSection.UserId);
      Console.WriteLine("Passwort: {0}", databaseSection.Password);
   }
   else
   {
      Console.WriteLine("Die Datenbankkonfiguration konnte nicht " +
         "eingelesen werden");
   }
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 155: Einlesen eigener Konfigurationsabschnitte in einer Konsolenanwendung

90 Anwendungskonfiguration über eine eigene XML-Datei

Die Anwendungskonfiguration über die Standard-Konfigurationsdatei einer Anwendung (siehe die vorhergehenden Rezepte) ist zwar recht einfach, besitzt aber leider auch einige Nachteile. Der wesentliche Nachteil ist, dass die Verwaltung der Daten in der Konfigurationsdatei für einen unbedarften Anwender u. U. etwas undurchsichtig ist.

Die im Folgenden beschriebene Klasse `Config` ermöglicht hingegen die Verwaltung von Konfigurationsdaten in einer einfachen XML-Datei, die beispielsweise folgendermaßen aussieht:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<config>
  <Database>
    <Server>Zaphod</Server>
    <UserId>Trillian</UserId>
    <Password>42</Password>
  </Database>
  <System>
    <StartTime>10:00</StartTime>
    <LastAccess>12.04.2003 14:03:36</LastAccess>
  </System>
</config>
```

Listing 156: Eine eigene Konfigurationsdatei

Die einzelnen Einstellungen sind in normalen XML-Elementen gespeichert und werden über übergeordnete Elemente in Sektionen eingeteilt. Eine Instanz der Klasse `Config` ermöglicht über eine Auflistung den Zugriff auf die einzelnen Sektionen. Innerhalb jeder Sektion steht eine weitere Auflistung für die Einstellungen zur Verfügung. Für jede zu speichernde Einstellung verwaltet die Klasse den Namen, den Wert, einen Defaultwert und eine Information darüber, ob die Einstellung beim Lesen der Konfigurationsdatei in dieser gefunden wurde.

Zur Realisation dieser Idee ist zunächst eine Klasse `Setting` zur Speicherung der einzelnen Einstellungen notwendig.

```
using System;
using System.IO;
using System.Xml;
using System.Collections.Generic;

...

public class Setting
{
   /* Der Name der Einstellung */
   public string Name;
```

Listing 157: Klasse zur Speicherung einer Einstellung

```
   /* Der Wert der Einstellung */
   public string Value;

   /* Der Defaultwert für das Lesen */
   public string DefaultValue;

   /* Gibt an, ob die Einstellung in der Datei gefunden wurde */
   public bool WasInFile;

   /* Konstruktor */
   public Setting(string name, string defaultValue)
   {
      this.Name = name;
      this.DefaultValue = defaultValue;
   }
}
```

Listing 157: Klasse zur Speicherung einer Einstellung (Forts.)

Die Klasse `Settings` implementiert eine Auflistung vom Typ `Dictionary` und ist spezialisiert auf Instanzen von `Setting`. `Settings` wird von der generischen `Dictionary`-Klasse, die als Grundlage für die Erzeugung von typsicheren assoziativen Auflistungen dient, abgeleitet. Die Methode `Add` dient dem Hinzufügen einer neuen Einstellung.

```
public class Settings : Dictionary<string, Setting>
{
   /* Methode zum Hinzufügen einer neuen Einstellung */
   public void Add(string settingName, string defaultValue)
   {
      this.Add(settingName,
         new Setting(settingName, defaultValue));
   }
}
```

Listing 158: Klasse zur Verwaltung eine Auflistung von Einstellungen

Die Klasse `Section` verwaltet die Daten einer Sektion. Neben dem Namen derselben enthält diese Klasse eine Eigenschaft `Settings` vom Typ der Einstellungs-Auflistung. In dieser Auflistung sollen die Einstellungen verwaltet werden, die der Sektion zugeordnet sind. Damit die Auflistung automatisch zur Verfügung steht, wird die Instanz der Klasse `Settings` im Konstruktor erzeugt:

```
public class Section
{
   /* Der Name der Sektion */
   public string Name;

   /* Die Einstellungen der Sektion */
   public Settings Settings;

   /* Konstruktor */
   public Section(string sectionName)
   {
      this.Name = sectionName;
      this.Settings = new Settings();
   }
}
```

Listing 159: Klasse zur Speicherung einer Einstellungs-Sektion

Ähnlich der Auflistung zur Verwaltung von Einstellungen implementiert die Klasse `Sections` eine `Dictionary`-Auflistung zur Verwaltung mehrerer `Section`-Instanzen:

```
public class Sections: Dictionary<string, Section>
{
   /* Fügt der Auflistung ein neues Section-Objekt hinzu */
   public void Add(string name)
   {
      this.Add(name, new Section(name));
   }
}
```

Listing 160: Auflistungs-Klasse zur Speicherung von Einstellungs-Sektionen

Die eigentliche Haupt-Klasse `Config` verwaltet in einer privaten Eigenschaft den Dateinamen der Konfigurationsdatei und in einer öffentlichen Eigenschaft eine Instanz der Klasse `Sections`. Im Konstruktor wird der Dateiname übergeben und die `Sections`-Instanz erzeugt:

```
public class Config
{
   /* Speichert den Dateinamen der XML-Datei */
   private string fileName;

   /* Verwaltet die Sektionen */
   public Sections Sections;
```

Listing 161: Eigenschaften und Konstruktor der Klasse Config

```
/* Konstruktor */
public Config(string fileName)
{
   this.fileName = fileName;
   this.Sections = new Sections();
}
```

Listing 161: Eigenschaften und Konstruktor der Klasse Config (Forts.)

Die Methode `Load` ermöglicht das Lesen der Einstellungen aus der Konfigurationsdatei. Diese Methode erzeugt dazu zunächst eine Instanz der Klasse `XmlDocument` und lädt die über das Feld `fileName` angegebene XML-Datei. Für den Fall, dass beim Laden ein Fehler auftritt, fängt die Methode die möglichen Ausnahmen ab und erzeugt daraus direkt eine neue Ausnahme, allerdings mit einer genaueren Fehlermeldung. Danach geht `Load` alle Sektionen und in diesen alle Einstellungen durch und ermittelt über einen XPath-Ausdruck das XML-Element, das diese Einstellung speichert. Wird das Element gefunden, liest die Methode dessen Wert aus und speichert ihn im aktuellen `Setting`-Objekt. Um später auswerten zu können, ob Einstellungen erfolgreich gelesen wurden, wird das Feld `WasInFile` des `Setting`-Objekts entsprechend mit `true` oder `false` beschrieben. `Load` gibt `true` zurück, wenn alle Einstellungen gelesen werden konnten. Konnte auch noch eine Einstellung nicht gelesen werden, gibt diese Methode `false` zurück.

```
public bool Load()
{
   // Variable für den Rückgabewert
   bool returnValue = true;

   // XmlDocument-Objekt für die Einstellungs-Datei erzeugen
   XmlDocument xmlDoc = new XmlDocument();

   // Datei laden
   try
   {
      xmlDoc.Load(this.fileName);
   }
   catch (IOException ex)
   {
      throw new IOException("Fehler beim Laden der Konfigurationsdatei '" +
         this.fileName + "': " + ex.Message);

   }
   catch (XmlException ex)
   {
      throw new XmlException("Fehler beim Laden der Konfigurationsdatei '" +
         this.fileName + "': " + ex.Message, ex);
   }
```

Listing 162: Methode zum Lesen der Konfigurationsdatei

```
      // Alle Sektionen durchgehen und die Einstellungen einlesen
      foreach (Section section in this.Sections.Values)
      {
         // Alle Einstellungen der Sektion durchlaufen
         foreach (Setting setting in section.Settings.Values)
         {
            // Einstellung im XML-Dokument lokalisieren
            XmlNode settingNode = xmlDoc.SelectSingleNode(
               "/config/" + section.Name + "/" + setting.Name);
            if (settingNode != null)
            {
               // Einstellung gefunden
               setting.Value = settingNode.InnerText;
               setting.WasInFile = true;
            }
            else
            {
               // Einstellung nicht gefunden
               setting.Value = setting.DefaultValue;
               setting.WasInFile = false;
               returnValue = false;
            }
         }
      }

      // Ergebnis zurückmelden
      return returnValue;
   }
```

Listing 162: Methode zum Lesen der Konfigurationsdatei (Forts.)

Die Methode `Save` speichert die Konfigurationsdaten in die XML-Datei. Sie erzeugt dazu eine neue Instanz der Klasse `XmlDocument` und initialisiert diese mit dem Skelett des XML-Dokuments (wobei als kleiner Trick die `LoadXml`-Methode mit einem String aufgerufen wird, der genau dieses Skelett beinhaltet).

Dann geht `Save` die einzelnen Sektionen der Einstellungen durch, um innerhalb des XML-Dokuments für jede Sektion und für jede Einstellung ein entsprechendes Element zu erzeugen. Schließlich wird die Datei noch gespeichert, wobei natürlich eventuelle Ausnahmen abgefangen und mit einer genaueren Fehlermeldung weitergegeben werden.

```
   public void Save()
   {
      // XmlDocument-Objekt für die Einstellungs-Datei erzeugen
      XmlDocument xmlDoc = new XmlDocument();
```

Listing 163: Methode zum Speichern der Konfigurationsdatei

```
      // Skelett der XML-Datei erzeugen
      xmlDoc.LoadXml("<?xml version=\"1.0\" encoding=\"utf-8\" " +
         "standalone=\"yes\"?><config></config>");

      // Alle Sektionen durchgehen und die Einstellungen schreiben
      foreach (Section section in this.Sections.Values)
      {
         // Element für die Sektion erzeugen und anfügen
         XmlElement sectionElement = xmlDoc.CreateElement(section.Name);
         xmlDoc.DocumentElement.AppendChild(sectionElement);

         // Alle Einstellungen der Sektion durchlaufen
         foreach (Setting setting in section.Settings.Values)
         {
            // Einstellungs-Element erzeugen und anfügen
            XmlElement settingElement =
               xmlDoc.CreateElement(setting.Name);
            settingElement.InnerText = setting.Value;
            sectionElement.AppendChild(settingElement);
         }
      }

      // Datei speichern
      try
      {
         xmlDoc.Save(this.fileName);
      }
      catch (IOException ex)
      {
         throw new IOException("Fehler beim Speichern der " +
            "Konfigurationsdatei '" + this.fileName + "': " + ex.Message);

      }
      catch (XmlException ex)
      {
         throw new XmlException("Fehler beim Speichern der " +
            " Konfigurationsdatei '" + this.fileName + "': " +
            ex.Message, ex);
      }
   }
}
```

Listing 163: Methode zum Speichern der Konfigurationsdatei (Forts.)

Die beispielhafte Anwendung dieser Klasse zeigt das folgende Listing:

```
// Config-Instanz erzeugen
Config config = new Config(Path.Combine(Application.StartupPath,
   "Config.xml"));

// Konfigurations-Sektionen und Einstellungen definieren
config.Sections.Add("System");
config.Sections.Add("Database");
config.Sections["System"].Settings.Add("LastAccess", "00:00");
config.Sections["Database"].Settings.Add("Server", "(local)");

// Datei einlesen
config.Load();

// Ein Datum lesen und ausgeben
Setting setting = config.Sections["System"].Settings["LastAccess"];
if (setting.WasInFile == false)
{
   Console.WriteLine("Einstellung System/LastAccess nicht gefunden");
}
Console.WriteLine("LastAccess: {0}", setting.Value);

// Den Wert ändern
setting.Value = DateTime.Now.ToString();

// Datei speichern
config.Save();
```

Listing 164: Anwendung der Klasse Config zur Verwaltung von Anwendungs-Konfigurationsdate

Hinweis

Die in diesem Rezept vorgestellten Klassen könnten noch wesentlich erweitert werden. So könnte die `Setting`-Klasse zusätzlich den Datentyp der Einstellung und bei numerischen Daten einen Minimal- und Maximalwert verwalten. Beim Einlesen bzw. Schreiben der Daten könnten diese dann auf den korrekten Datentyp und die Einhaltung der Wertgrenzen überprüft werden. Eine weitere sinnvolle Idee ist die automatische Erzeugung eines Formulars über eine weitere Methode, in dem der Anwender die Einstellungen bearbeiten kann. Das Formular könnte die einzelnen Sektionen und deren Einstellungen in einem `TreeView`-Steuerelement darstellen. Wählt der Anwender eine Einstellung aus, wird diese in einem separaten Bereich des Formulars zur Bearbeitung in einer `TextBox` dargestellt.

91 Daten aus der Registry lesen und in die Registry schreiben

Über die Klasse `Registry` aus dem Namensraum `Microsoft.Win32` können Sie Daten aus der Windows-Registry lesen, Daten manipulieren, neue Daten in vorhandenen Registry-Schlüsseln erzeugen und neue Schlüssel erzeugen.

Basis für die Arbeit mit der Registry ist die Klasse `RegistryKey`, die einen (Unter-) Schlüssel der Registry repräsentiert. Über verschiedene Methoden können Sie den Schlüssel bearbeiten. Die Methode `GetValue` liest z. B. den Wert eines Eintrags in einem Schlüssel, über die Methode `SetValue` können Sie einen Wert schreiben. Die Methode `CreateSubKey` ermöglicht das Erzeugen eines Unterschlüssels, über `DeleteSubKey` können Sie einen Unterschlüssel löschen. Die `OpenSubKey`-Methode öffnet einen Schlüssel, der einem anderen Schlüssel untergeordnet ist.

Die Wurzel-Schlüssel der Registry erreichen Sie über die statischen Eigenschaften `ClassesRoot (HKEY_CLASSES_ROOT)`, `CurrentConfig (HKEY_CURRENT_CONFIG)`, `CurrentUser (HKEY_CURRENT_USER)`, `DynData (HKEY_DYN_DATA)`, `LocalMachine (HKEY_LOCAL_MACHINE)`, `PerformanceData (HKEY_PERFORMANCE_DATA)` und `Users` (`HKEY_USERS`). Da diese Eigenschaften ein `RegistryKey`-Objekt referenzieren, können Sie über die `OpenSubKey`-Methode zu einem Unterschlüssel navigieren. Die einzelnen Schlüsselnamen des (relativen) Pfads geben Sie durch Backslashes getrennt an.

Das folgende Beispiel liest den Wert *DOC-PATH* des Schlüssels *HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Word\Options*. Das Programm benötigt den Import der Namensräume `System` und `Microsoft.Win32`.

```
RegistryKey regKey = Registry.CurrentUser.OpenSubKey(
   @"Software\Microsoft\Office\11.0\Word\Options");

string wordDocPath = null;
if (regKey != null)
{
    object regValue = regKey.GetValue("DOC-PATH");
    if (regValue != null)
    {
        wordDocPath = regValue.ToString();
        Console.WriteLine(wordDocPath);
    }
    else
    {
        Console.WriteLine("Wert nicht gefunden");
    }
}
else
{
    Console.WriteLine("Schlüssel nicht gefunden");
}
```

Listing 165: Lesen eines Registry-Eintrags

Beim Schreiben sollten Sie für jeden Unterschlüssel überprüfen, ob dieser existiert, und den Schlüssel gegebenenfalls anlegen, bis Sie am letzten Schlüssel angelangt sind und dort den Wert schreiben können. Da das Ganze für die Praxis ein wenig kompliziert ist, habe ich eine Klasse `RegistryUtils` entwickelt, die den Registry-Zugriff erleichtert.

Den Methoden dieser Klasse wird der Registry-Pfad in der unter Windows üblichen Form übergeben, wobei der Wurzel-Schlüssel allerdings separat übergeben wird. Ein solcher Pfad sieht dann z. B. so aus:

Software\Microsoft\Office\10.0\Word\Options\DOC-PATH

Um den Wurzel-Schlüssel angeben zu können, habe ich zunächst eine Aufzählung implementiert:

```
public enum RegistryRootKeys
{
    HKEY_CLASSES_ROOT,
    HKEY_CURRENT_CONFIG,
    HKEY_CURRENT_USER,
    HKEY_DYN_DATA,
    HKEY_LOCAL_MACHINE,
    HKEY_PERFORMANCE_DATA,
    HKEY_USERS
}
```

Listing 166: Aufzählung zur Angabe der Registry-Wurzel-Schlüssel

Die Methoden der Klasse `RegistryUtils` fangen mögliche Ausnahmen (z. B. die Ausnahme vom Typ `UnauthorizedAccessException` bei einem unautorisierten Zugriff) nicht ab, sodass diese an den Aufrufer weitergereicht werden.

Die erste (private) Methode `GetRegistryRootKey` gibt eine Referenz auf das zu einem Wert der `RegistryRootKeys`-Aufzählung passende `RegistryKey`-Objekt zurück. Diese Methode wird in den anderen Methoden der Klasse aufgerufen.

```
public class RegistryUtils
{
   private static RegistryKey GetRegistryRootKey(RegistryRootKeys rootKey)
   {
      // Ermitteln des Registry-Wurzel-Schlüssels
      RegistryKey regKey = null;
      switch (rootKey)
      {
         case RegistryRootKeys.HKEY_CLASSES_ROOT:
            regKey = Registry.ClassesRoot;
            break;
```

Listing 167: Methode zum Ermitteln einer Referenz auf ein Wurzel-RegistryKey-Objekt

```
            case RegistryRootKeys.HKEY_CURRENT_CONFIG:
               regKey = Registry.CurrentConfig;
               break;

            case RegistryRootKeys.HKEY_CURRENT_USER:
               regKey = Registry.CurrentUser;
               break;

            case RegistryRootKeys.HKEY_DYN_DATA:
               regKey = Registry.DynData;
               break;

            case RegistryRootKeys.HKEY_LOCAL_MACHINE:
               regKey = Registry.LocalMachine;
               break;

            case RegistryRootKeys.HKEY_PERFORMANCE_DATA:
               regKey = Registry.PerformanceData;
               break;

            case RegistryRootKeys.HKEY_USERS:
               regKey = Registry.Users;
               break;
         }

         return regKey;
      }
```

Listing 167: Methode zum Ermitteln einer Referenz auf ein Wurzel-RegistryKey-Objekt (Forts.)

Die Methode zum Lesen eines Registry-Eintrags (`ReadValue`) ruft zunächst `GetRegistryRootKey` auf um das zum übergebenen Wurzel-Schlüssel-Wert passende `RegistryKey`-Objekt zu referenzieren. Über diese Referenz ruft `ReadValue` dann die `OpenSubKey`-Methode unter der Übergabe des Registry-Pfades auf. Wird der Schlüssel gefunden, liest die Methode den Wert des übergebenen Wertnamens. Im Erfolgsfall liefert `ReadValue` den gelesenen Wert zurück. Wird der Wert nicht gefunden, gibt diese Methode den übergebenen Defaultwert zurück.

```
public static object ReadValue(RegistryRootKeys rootKey, string keyPath,
   string valueName, object defaultValue)
{
   // Schlüssel ermitteln
   RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(keyPath);
   // Wert auslesen, wenn der Unterschlüssel gefunden wurde
   object regValue = defaultValue;
   if (regKey != null)
```

Listing 168: Methode zum Lesen eines Registry-Eintrags

```
    {
        regValue = regKey.GetValue(valueName);
    }

    return regValue;
}
```

Listing 168: Methode zum Lesen eines Registry-Eintrags (Forts.)

`ReadValue` gibt (entsprechend der `GetValue`-Methode des `RegistryKey`-Objekts) einen String zurück, wenn der Registry-Eintrag vom Typ `REG_SZ` oder `REG_EXPAND_SZ` ist. Bei einem mehrteiligen String (`REG_MULTI_SZ`) wird ein String-Array mit den einzelnen Teilen zurückgegeben. Ein numerischer Wert (`REG_DWORD`) wird als `int` zurückgegeben, binäre Daten (`REG_BINARY`) als Byte-Array.

Ich habe bewusst bei dieser Methode auf die Erzeugung einer Ausnahme für den Fall, dass der Eintrag nicht gefunden wurde, verzichtet. Durch die Übergabe eines geeigneten Defaultwerts können Sie recht flexibel auf das Nichtvorhandensein eines Eintrags reagieren. Übergeben Sie beim Aufruf dieser Methode als Defaultwert `null`, können Sie problemlos entscheiden, ob der Eintrag gefunden wurde. Gibt `ReadValue` in diesem Fall `null` zurück, wurde der Eintrag nicht gefunden (für leere, aber vorhandene Einträge gibt `ReadValue` je nach Datentyp des Eintrags einen Leerstring, 0 oder ein leeres Array zurück).

Die Methode zum Schreiben von Einträgen (`WriteValue`) arbeitet ähnlich. Übergeben wird wie bei `ReadValue` ein Wert der Aufzählung `RegistryRootKeys`, der den Wurzel-Schlüssel bezeichnet, der Pfad des Schlüssels und der Name des Eintrags. Dann folgen der Wert des Eintrags und eine Information darüber, ob `WriteValue` Unterschlüssel anlegen soll, falls diese nicht gefunden wurden.

Abhängig vom Typ des übergebenen zu schreibenden Wertes erzeugt die in `WriteValue` verwendete `SetValue`-Methode einen Eintrag vom Typ `REG_SZ`, `REG_DWORD` oder `REG_BINARY`. Ein `REG_BINARY`-Eintrag wird erzeugt, wenn Sie ein Byte-Array übergeben. Ein `REG_DWORD`-Wert wird erzeugt, wenn Sie einen `unit`- oder `int`-Wert übergeben. In allen anderen Fällen wird ein `REG_SZ`-Wert erzeugt.

`WriteValue` ermittelt zuerst eine Referenz auf das Wurzel-`RegistryKey`-Element und zerlegt den Pfad dann in seine Teile. Beim Durchgehen des Pfads wird ein nicht existierender Unterschlüssel über die `CreateSubKey`-Methode angelegt, falls das Argument `createIfNotExist` `true` ist. Der jeweilige Schlüssel muss dazu durch die Übergabe von `true` am zweiten Argument der `OpenSubKey`-Methode so geöffnet werden, dass auch ein Schreiben möglich ist. Schließlich wird noch der Wert geschrieben, wenn der letzte Unterschlüssel ermittelt werden konnte. Für den Fall, dass dieser nicht gefunden wird, erzeugt `WriteValue` eine Ausnahme.

```
public static void WriteValue(RegistryRootKeys rootKey, string keyPath,
    string valueName, object value, bool createIfNotExist)
{
```

Listing 169: Methode zum Schreiben eines Registry-Eintrags

```
      // Wurzel-Schlüssel ermitteln
      RegistryKey regKey = GetRegistryRootKey(rootKey);

      // Den Pfad in seine Einzelteile zerlegen
      string[] pathToken = keyPath.Split('\\');

      // Pfad durchgehen und den Schlüssel referenzieren
      RegistryKey subKey = null;
      for (int i = 0; i < pathToken.Length; i++)
      {
         if (regKey != null)
         {
            // Unterschlüssel zum Lesen und Schreiben öffnen
            subKey = regKey.OpenSubKey(pathToken[i], true);

            if (subKey == null && createIfNotExist)
            {
               // Unterschlüssel nicht gefunden: Schlüssel erzeugen
               // falls dies gewünscht ist
               subKey = regKey.CreateSubKey(pathToken[i]);
            }
            regKey = subKey;
         }
      }

      // Wert schreiben, wenn der Unterschlüssel gefunden wurde
      if (regKey != null)
      {
         regKey.SetValue(valueName, value);
      }
      else
      {
         // Ausnahme werfen
         throw new Exception("Schlüssel " + rootKey.ToString() + "\\" +
            keyPath + " nicht gefunden");
      }
   }
```

Listing 169: Methode zum Schreiben eines Registry-Eintrags (Forts.)

Nun fehlt noch eine Methode zum Löschen eines Eintrags. `DeleteValue` referenziert dazu zunächst den Schlüssel, der dem übergebenen Pfad entspricht. Wird der Schlüssel gefunden, ruft `DeleteValue` dessen `Delete`-Methode. Im negativen Fall erzeugt die Methode eine Ausnahme:

```
public static void DeleteValue(RegistryRootKeys rootKey, string keyPath,
   string valueName)
{
   // Unterschlüssel zum Schreiben öffnen
   RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(
      keyPath, true);
   // Wert über den übergeordneten Schlüssel löschen
   // wenn dieser gefunden wurde
   if (regKey != null)
   {
      regKey.DeleteValue(valueName, false);
   }
   else
   {
      // Schlüssel nicht gefunden: Ausnahme werfen
      throw new Exception("Schlüssel " + rootKey.ToString() + "\\" +
         keyPath + " nicht gefunden");
   }
}
```

Listing 170: Methode zum Löschen eines Werts in der Registry

Die Methode `DeleteKey` löscht schließlich einen ganzen Schlüssel. Da ein Schlüssel über die `DeleteSubKey`-Methode des übergeordneten Schlüssels gelöscht werden muss, zerlegt `DeleteKey` den übergebenen Pfad zunächst in den Pfad zum übergeordneten Schlüssel und den Namen des zu löschenden Schlüssels. Mit dem so ermittelten Pfad referenziert die Methode den übergeordneten Schlüssel und ruft dessen `DeleteSubKey`-Methode mit dem Namen des Unterschlüssels auf. Für den Fall, dass der übergeordnete Schlüssel nicht gefunden wurde, erzeugt `DeleteKey` eine Ausnahme:

```
public static void DeleteKey(RegistryRootKeys rootKey, string keyPath)
{
   // Den Pfad zum übergeordneten Schlüssel und den Namen des zu
   // löschenden Schlüssels ermitteln
   int i = keyPath.LastIndexOf("\\");
   string parentKeyPath = keyPath.Substring(0, i);
   string keyName = keyPath.Substring(i + 1, keyPath.Length - i - 1);

   // Den dem zu löschenden Schlüssel übergeordneten Schlüssel zum
   // Schreiben öffnen
   RegistryKey regKey = GetRegistryRootKey(rootKey).OpenSubKey(
      parentKeyPath, true);

   if (regKey != null)
   {
```

Listing 171: Methode zum Löschen eines Schlüssels in der Registry

```
            // Schlüssel über den übergeordneten Schlüssel löschen
            regKey.DeleteSubKey(keyName);
         }
         else
         {
            // Schlüssel nicht gefunden: Ausnahme werfen
            throw new Exception("Schlüssel " + rootKey.ToString() + "\\" +
               keyPath + " nicht gefunden");
         }
      }
   }
```

Listing 171: Methode zum Löschen eines Schlüssels in der Registry (Forts.)

Damit ist die Klasse fertig. Listing 172 zeigt eine beispielhafte Anwendung dieser Klasse.

```
// Wert des Eintrags HKEY_CURRENT_USER\Software\Microsoft\Office\
// 11.0\Word\Options\DOC-PATH lesen
object wordDocPath = RegistryUtils.ReadValue(
   RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Microsoft\Office\11.0\Word\Options", "DOC-PATH", null);
if (wordDocPath != null)
{
   Console.WriteLine(wordDocPath);
}
else
{
   Console.WriteLine("Eintrag nicht gefunden");
}

// Eintrag
// HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Version
// so schreiben, dass der Eintrag inklusive allen Schlüsseln
// automatisch angelegt wird, falls er noch nicht existiert
RegistryUtils.WriteValue(RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Addison-Wesley\Codebook", "Version", "1.0", true);

// Eintrag
// HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Samples schreiben
RegistryUtils.WriteValue(RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Addison-Wesley\Codebook\Samples", "Registry",
   "c:\\Samples\\Registry", true);
```

Listing 172: Anwendung der Klasse RegistryUtils

```
// Schlüssel
// HKEY_CURRENT_USER\Software\Addison-Wesley\Codebook\Samples löschen
RegistryUtils.DeleteKey(RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Addison-Wesley\Codebook\Samples");

// DWord-Wert schreiben
RegistryUtils.WriteValue(RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Addison-Wesley\Codebook", "DWord-Test-Value", 123, true);

// Binary-Wert schreiben
byte[] data = new byte[255];
for (byte i = 0; i < 255; i++)
{
   data[i] = i;
}
RegistryUtils.WriteValue(RegistryRootKeys.HKEY_CURRENT_USER,
   @"Software\Addison-Wesley\Codebook", "Binary-Test-Value", data, true);
```

Listing 172: Anwendung der Klasse RegistryUtils (Forts.)

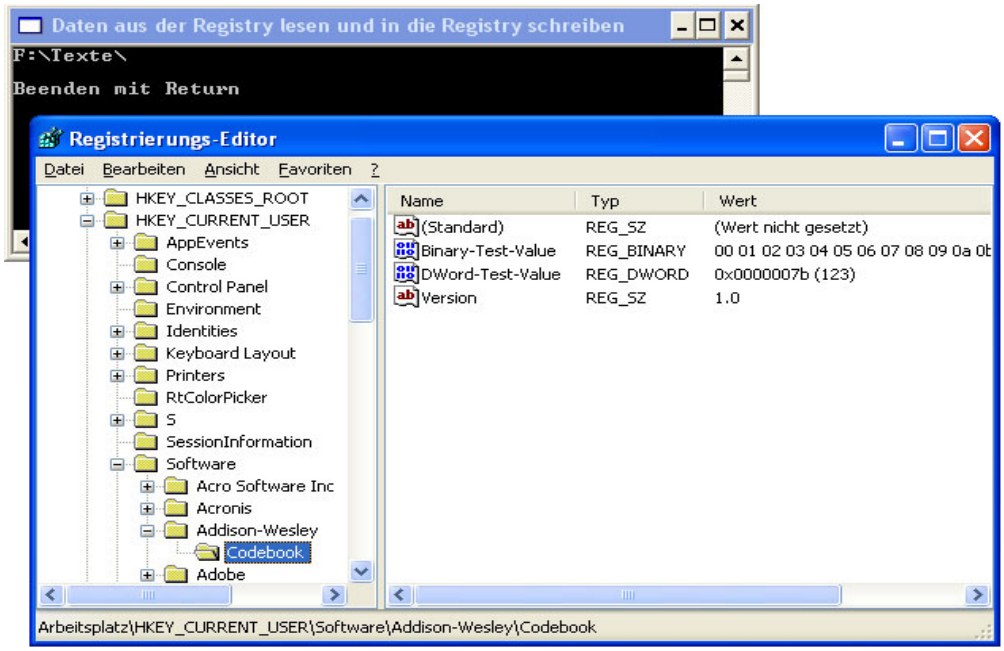


Abbildung 58: Das Beispielprogramm zum Lesen und Schreiben der Registry und der Registrierungs-Editor mit dem erzeugten Registry-Schlüssel

92 Verhindern, dass eine Anwendung mehrfach gestartet werden kann

In einigen Fällen ist es sinnvoll zu verhindern, dass eine Anwendung mehrfach gestartet werden kann. Das ist z. B. dann der Fall, wenn die Anwendung sehr langsam startet (weil beim Start umfangreiche Ressourcen geöffnet werden) oder wenn die Anwendung beim Start Ressourcen wie z. B. Dateien exklusiv öffnet (die dann beim zweiten Start nicht mehr geöffnet werden können bzw. dürfen).

Den mehrfachen Start können Sie auf verschiedene Weise verhindern. Einmal können Sie (idealerweise in der `Main`-Funktion) abfragen, ob bereits ein Prozess gleichen Namens wie die Anwendung ausgeführt wird. Dazu können Sie die `GetProcessesByName`-Methode der Klasse `System.Diagnostics.Process` einsetzen. Dieser Methode übergeben Sie den Namen des aktuellen Prozesses, den Sie über `System.Diagnostics.Process.GetCurrentProcess().ProcessName` ermitteln. `GetProcessesByName` gibt ein Array von `Process`-Objekten zurück, das alle aktuell laufenden Prozesse repräsentiert, die den angegebenen Namen tragen. Da der aktuelle Prozess mit aufgeführt wird, könnten Sie die Länge des Arrays mit > 1 vergleichen um herauszufinden, ob mehr als ein Prozess mit dem Namen des aktuellen Prozesses ausgeführt wird.

Leider ist dieses Vorgehen nicht sicher. Der Prozessname entspricht nämlich dem Namen der Assembly (also dem Namen der ausführbaren Datei). Wird gerade eine andere Anwendung mit demselben Dateinamen ausgeführt, wird dieser Prozess auch aufgeführt.

Als Lösung können Sie die Prozesse durchgehen, die denselben Namen tragen wie der aktuelle, und zusätzlich den Fenstertitel vergleichen. Um den aktuellen Prozess auszuschließen, vergleichen Sie die Prozess-Id der Prozesse. Dieses Vorgehen schließt jedoch nicht aus, dass es unter Umständen auch einen Prozess einer eigentlich komplett anderen Anwendung geben kann, der denselben Dateinamen und zufällig auch denselben Fenstertitel trägt. Außerdem funktioniert diese Technik nicht, wenn Sie den Fenstertitel des Hauptfensters beim Start der Anwendung dynamisch setzen.

Alternativ können Sie die gefundenen Prozesse daraufhin überprüfen, ob die ausführbare Datei in demselben Ordner gespeichert ist, womit Sie ausschließen, dass versehentlich eine fremde Anwendung als die eigene Anwendung erkannt wird. Dummerweise ist es dann möglich die eigene Anwendung aus verschiedenen Ordnern heraus mehrfach zu starten.

Die scheinbar einzig vernünftige Lösung für diese Probleme ist die Verwendung eines *Mutex*. Ein Mutex ist ein Systemobjekt, das eigentlich für das (prozessübergreifende) Synchronisieren von Threads verwendet wird. Bei der Erzeugung eines Mutex können Sie diesem einen systemweiten Namen zuweisen.

Ein Thread kann über die `WaitOne`-Methode eines `Mutex`-Objekts dieses in einen »signalisierten« Zustand versetzen. Ist das Objekt gerade in diesem Zustand, wartet `WaitOne` so lange, bis der andere Thread den Mutex über die `ReleaseMutex`-Methode freigegeben hat (oder bis zu einem definierten Timeout). `WaitOne` gibt `true` zurück, wenn der Mutex in den Besitz des Thread gelangt ist. Nach dem erfolgreichen Aufruf von `WaitOne` kann der Thread dann auf externe Ressourcen zugreifen, ohne die Gefahr einzugehen, dass andere Threads, die denselben Mutex verwenden, gleichzeitig auch darauf zugreifen. Der Mutex wird schließlich über `ReleaseMutex` wieder freigegeben, damit andere Threads diesen über `WaitOne` sperren können.

Um den Mehrfachstart einer Anwendung zu verhindern, können Sie einen Mutex zweckentfremden. Dazu erzeugen Sie ein `Mutex`-Objekt, dem Sie im Konstruktor am zweiten Argument einen systemweit eindeutigen Namen übergeben. Die Methode `IsOtherInstanceRunning` in Listing 173 verwendet als Namen den Anwendungsnamen mit dem Zusatz »_MultiStartPrevent«. Am ersten Argument können Sie festlegen, ob der Mutex direkt in den Besitz des Thread gelangen soll (falls dies möglich ist). Da dies in der Praxis wenig Sinn macht (weil Sie dazu auch `WaitOne` aufrufen können), übergibt `IsOtherInstanceRunning` an diesem Argument `false`.

Dann können Sie über `WaitOne` den Besitz des Mutex anfordern. `IsOtherInstanceRunning` übergibt am ersten Argument als Timeout 0, damit `WaitOne` direkt zurückkehrt. Am zweiten Argument wird `true` übergeben, was hier keine weitere Bedeutung besitzt. Über dieses Argument können Sie bei Anwendungen, die vor dem Warten auf den Mutex den Zugriff auf Objekte oder Klassen über `lock` (bzw. über die `Monitor`-Klasse) sperren, festlegen, dass die Sperrung vor dem Warten aufgehoben und nach dem Warten wieder gesetzt wird. Damit verhindern Sie Deadlocks, wenn Sie gleichzeitig über `lock` sperren und Mutexe verwenden.

`IsOtherInstanceRunning` fragt dann einfach die Rückgabe von `WaitOne` ab um zu entscheiden, ob der Mutex in den Besitz des aktuellen Thread gelangt ist. Der Mutex wird dann auch nicht explizit freigegeben, damit dieser so lange signalisiert bleibt, wie die Anwendung geöffnet ist. Beim Beenden der Anwendung wird der Mutex automatisch freigegeben.

Zum Kompilieren der Methode müssen Sie die Namensräume `System` und `System.Threading` einbinden.

```
/* Variable für den Mutex, der von der Anwendung erzeugt wird */
private static Mutex mutex;

/* Methode zur Ermittlung, ob bereits eine Instanz der Anwendung
   ausgeführt wird */
public static bool IsOtherInstanceRunning()
{
   // Mutex mit einem systemweit eindeutigen Namen erzeugen
   string mutexName = System.Windows.Forms.Application.ProductName +
      "_MultiStartPrevent";
   mutex = new Mutex(false, mutexName);

   // Signalisieren des Mutex und gleichzeitig abfragen, ob bereits ein
   // gleichnamiger Mutex existiert
   if (mutex.WaitOne(0, true))
   {
      return false;
   }
   else
   {
      return true;
   }
}
```

Listing 173: Mehrfachstart einer Anwendung über einen Mutex verhindern

Die `IsOtherInstanceRunning`-Methode können Sie nun auch in der `Main`-Methode einsetzen:

```
static void Main()
{
   // Überprüfen, ob die Anwendung bereits ausgeführt wird
   if (IsOtherInstanceRunning())
   {
      // Die Anwendung wird bereits ausgeführt, also eine Meldung ausgeben
      // und die Anwendung beenden
      MessageBox.Show("Dieses Programm kann nicht mehrfach ausgeführt " +
         "werden.", Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Exclamation);
   }
   else
   {
      /* Anwendung mit dem Startformular starten */
      Application.EnableVisualStyles();
      Application.Run(new StartForm());
   }
}
```

Listing 174: Verhindern des Mehrfachstarts über IsOtherInstanceRunning

93 Aktivieren einer laufenden Anwendung

Wenn Sie eine laufende Anwendung aktivieren wollen, was besonders nach dem Verhindern des mehrfachen Starts einer Anwendung sinnvoll sein kann, könnten Sie die zu Visual Basic gehörende Methode `AppActivate` des `Interaction`-Moduls aus dem Namensraum `Microsoft.VisualBasic` verwenden. Leider ist `AppActivate` nicht in der Lage, eine minimierte Anwendung nach dem Aktivieren in den normalen Modus wiederherzustellen. Die Anwendung wird zwar aktiviert, bleibt aber minimiert. Deshalb verwende ich für die Lösung des Problems Windows-API-Funktionen.

Zum Kompilieren des Quellcodes müssen Sie zunächst die Namensräume `System`, `System.Text`, `System.Diagnostics` und `System.Runtime.InteropServices` importieren und die benötigten API-Funktionen deklarieren:

```
[DllImport("user32", SetLastError=true)]
private static extern int FindWindow(string lpClassName, string lpWindowName);

[DllImport("user32", SetLastError = true)]
private static extern int GetWindowTextLength(IntPtr hWnd);

[DllImport("user32", SetLastError = true)]
```

Listing 175: Deklaration der benötigten API-Funktionen

```
private static extern int GetWindowText(IntPtr hWnd, StringBuilder lpString, int
nMaxCount);

[DllImport("user32", SetLastError=true)]
public static extern int SetForegroundWindow(IntPtr hwnd);

[DllImport("user32", SetLastError=true)]
public static extern int IsIconic(IntPtr hwnd);

[DllImport("user32", SetLastError=true)]
public static extern int OpenIcon(IntPtr hwnd);

[DllImport("user32", SetLastError=true)]
private static extern int GetClassName(IntPtr hWnd,
   StringBuilder lpClassName, int nMaxCount);
```

Listing 175: Deklaration der benötigten API-Funktionen (Forts.)

Aktivieren über den genauen Fenstertitel

Die Basis-Methode `ActivateApplicationByCompleteTitle` aktiviert ein Fenster über den genauen Fenstertitel und (optional) den Klassennamen der Fenster-Klasse.

Diese Methode sucht zunächst über die Funktion `FindWindow` ein Fenster, das den übergebenen Fenstertitel und/oder Klassennamen trägt. Der Klassenname ist der Windows-interne Name der Klasse des Fensters. Wenn Sie für den Klassennamen `null` übergeben, sucht `FindWindow` ein beliebiges Fenster mit dem übergebenen Titel. Ich habe den Klassennamen mit in `ActivateApplicationByCompleteTitle` aufgenommen, da es möglich ist, dass zwei verschiedene Fenster denselben Titel tragen. Über den Klassennamen können Sie das Fenster bei Bedarf genauer spezifizieren. Um diesen herauszufinden, können Sie ein Tool wie das Microsoft-Werkzeug *Spy* verwenden (das ein Teil des Plattform-SDK ist). Um das Ganze zu vereinfachen, habe ich aber auch eine Hilfsmethode `GetWindowClassName` entwickelt, der ein Fenstertitel übergeben wird und die den Klassennamen zurückgibt.

`FindWindow` gibt den Handle des Fensters zurück, wenn eines gefunden wurde. In diesem Fall überprüft `ActivateApplicationByCompleteTitle` über die API-Funktion `IsIconic`, ob das Fenster minimiert ist. Ist dies der Fall, wird das Fenster über `OpenIcon` wiederhergestellt. Danach ruft `ActivateApplicationByCompleteTitle` die API-Funktion `SetForegroundWindow` auf um das Fenster in den Vordergrund zu holen.

```
/* Aktiviert eine Anwendung über den kompletten Titel des Hauptfensters */
public static void ActivateApplicationByCompleteTitle(
   string mainWindowTitle, string mainWindowClassName)
{
   // Fenster mit dem übergebenen Klassennamen und Titel suchen
   int windowHandle = FindWindow(mainWindowClassName, mainWindowTitle);
```

Listing 176: Methoden zum Aktivieren einer Anwendung und zum Auslesen des Klassennamens eines Fensters

```
   if (windowHandle != 0)
   {
      // Fenster gefunden: Überprüfen, ob das Fenster verkleinert ist
      if (IsIconic((IntPtr)windowHandle) != 0)
      {
         // Fenster wiederherstellen
         if (OpenIcon((IntPtr)windowHandle) == 0)
         {
            throw new Exception("Windows-Fehler " +
               Marshal.GetLastWin32Error() + " beim Wiederherstellen " +
               "des Fensters mit dem Titel '" + mainWindowTitle +
               "' aus dem minimierten Zustand.");
         }
      }

      // Fenster in den Vordergrund holen
      if (SetForegroundWindow((IntPtr)windowHandle) == 0)
      {
         throw new Exception("Windows-Fehler " +
            Marshal.GetLastWin32Error() + " beim Nach-Vorne-Holen " +
            "des Fensters mit dem Titel '" + mainWindowTitle + "'");
      }
   }
   else
   {
      throw new Exception("Fenster mit Titel '" + mainWindowTitle +
         "' nicht gefunden");
   }
}

/* Hilfs-Methode zum Ermitteln des Klassennamens eines Fensters */
public static string GetWindowClassName(string windowTitle)
{
   int windowHandle = FindWindow(null, windowTitle);
   if (windowHandle != 0)
   {
      StringBuilder className = new StringBuilder(255);
      if (GetClassName((IntPtr)windowHandle, className, 255) > 0)
      {
         return className.ToString();
      }
      else
      {
         throw new Exception("Windows-Fehler " +
            Marshal.GetLastWin32Error() + "beim Ermitteln des " +
```

Listing 176: Methoden zum Aktivieren einer Anwendung und zum Auslesen des Klassennamens eines Fensters (Forts.)

```
            " Klassennamens des Fensters mit dem Titel '" +
            windowTitle + "'");
      }
   }
   else
   {
      throw new Exception("Fenster mit dem Titel '" + windowTitle +
         "' nicht gefunden");
   }
}
```

Listing 176: Methoden zum Aktivieren einer Anwendung und zum Auslesen des Klassennamens eines Fensters (Forts.)

Alle verwendeten API-Funktionen liefern im Fehlerfall 0 zurück. In diesen Fällen generieren `ActivateApplicationByCompleteTitle` und `GetWindowClassName` eine Ausnahme. Um den zugrunde liegenden Windows-Fehler zu ermitteln, rufen beide Methoden `Marshal.GetLastWin32Error` auf, die den zuletzt aufgetretenen Windows-Fehler ausliest. Ich habe bewusst darauf verzichtet, den Windows-Fehlercode über die API-Funktion `FormatMessage` in eine lesbare Fehlermeldung umzuwandeln, um die Methoden übersichtlich zu gestalten und weil Windows-Fehler bei der Anwendung der eingesetzten Funktionen wohl eher selten auftreten.

Zur Vervollständigung habe ich eine weitere Variante von `ActivateApplicationByCompleteTitle` implementiert, die nur den Fenstertitel übergeben bekommt:

```
public static void ActivateApplicationByCompleteTitle(
   string mainWindowTitle)
{
   ActivateApplicationByCompleteTitle(mainWindowTitle, null)
}
```

Listing 177: Methode zum Aktivieren einer Anwendung über den Fenstertitel, ohne den Klassennamen anzugeben

Aktivieren über einen teilweise angegebenen Fenstertitel

Da viele Anwendungen im Titel neben ihrem Namen zusätzliche Informationen ausgeben, ist es in vielen Fällen nicht möglich, diese über den kompletten Titel zu aktivieren. Deshalb habe ich zwei weitere Methoden (`ActivateApplicationByPartialTitle`) geschrieben, denen lediglich ein Teil des Titels übergeben werden kann.

Die Basisversion dieser Methoden geht alle Prozesse durch und ermittelt, ob diese ein Hauptfenster besitzen. Ist dies der Fall, liest die Methode über die API-Funktionen `GetWindowTextLength` und `GetWindowText` den Titel des Hauptfensters aus und vergleicht diesen mit dem übergebenen Fenstertitel. Dabei wird nur überprüft, ob der übergebene String in dem Fenstertitel vorkommt. Wird der Vergleich wahr, ruft `ActivateApplicationByPartialTitle` die Methode `ActivateApplicationByCompleteTitle` auf, um die Anwendung zu aktivieren.

Der Basisversion von `ActivateApplicationByPartialTitle` wird neben dem Fenstertitel auch der Klassenname übergeben um das Fenster genauer spezifizieren zu können. Eine zweite Variante dieser Methode besitzt lediglich ein Argument für den Fenstertitel für den Fall, dass Sie den Klassennamen nicht kennen:

```
/* Aktiviert eine Anwendung über den teilweise angegebenen
  Titel des Hauptfensters */
public static void ActivateApplicationByPartialTitle(
   string mainWindowTitle, string mainWindowClassName)
{
   string errors = null;
   int count = 0;
   // Alle Prozesse durchgehen
   foreach (Process process in Process.GetProcesses())
   {
      if ((int)process.MainWindowHandle != 0)
      {
         // Ermitteln der Länge des Textes des Hauptfensters
         int windowTextLength =
            GetWindowTextLength(process.MainWindowHandle);

         // Einlesen des Textes des Hauptfensters
         StringBuilder windowTextBuilder = new StringBuilder(
            windowTextLength + 1);
         if (GetWindowText(process.MainWindowHandle,
            windowTextBuilder, windowTextLength + 1) > 0)
         {
            // Ermitteln, ob der übergebene Titel in dem Text
            // des aktuellen Fensters vorkommt und aktivieren
            // dieses Fensters im positiven Fall
            string windowText = windowTextBuilder.ToString();
            if (windowText.IndexOf(mainWindowTitle) > -1)
            {
               try
               {
                  ApplicationUtils.ActivateApplicationByCompleteTitle(
                     windowText, mainWindowClassName);
                  count++;
               }
               catch (Exception ex)
               {
                  if (errors != null)
                  {
                     errors += Environment.NewLine;
```

Listing 178: Methoden zum Aktivieren einer Anwendung über einen teilweise angegebenen Titel

```
                }
                errors += ex.Message;
            }
        }
    }
  }

  // Fehler auswerten
  if (errors != null)
  {
    throw new Exception(errors);
  }
  else if (count == 0)
  {
    // Es wurde kein Fenster mit dem übergebenen Titel
    // gefunden
    throw new Exception("Fenster mit Titel '" + mainWindowTitle +
      "' nicht gefunden");
  }
}

/* Aktiviert eine Anwendung über den teilweise angegebenen
   Titel des Hauptfensters */
public static void ActivateApplicationByPartialTitle(
  string mainWindowTitle)
{
  ActivateApplicationByPartialTitle(mainWindowTitle, null);
}
```

Listing 178: Methoden zum Aktivieren einer Anwendung über einen teilweise angegebenen Titel (Forts.)

Hinweis

Da `ActivateApplicationByPartialTitle` alle laufenden Prozesse durchgeht, werden natürlich auch alle Anwendungen aktiviert, die den übergebenen Text im Titel tragen und deren Hauptfenster gegebenenfalls den übergebenen Klassennamen aufweist. Wenn Sie dies ausprobieren, sollten Sie jedoch beachten, dass viele Anwendungen, die scheinbar in mehreren Instanzen ausgeführt werden, nur einen Prozess verwenden. Microsoft Word und der Mozilla Firefox sind Beispiele dafür. Wenn Sie eine solche Anwendung aktivieren, wird nur eine Instanz in den Vordergrund geholt.

Das folgende Listing zeigt eine Anwendung der `ActivateApplicationByPartialTitle`-Methode, bei der das Fenster des Mozilla-Firefox-Browsers aktiviert wird:

```
try
{
   ApplicationUtils.ActivateApplicationByPartialTitle(
      "Mozilla Firefox", "MozillaWindowClass");
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 179: Beispiel-Anwendung der ActivateApplicationByPartialTitle-Methode

Aktivieren über ein Process-Objekt

Wenn Sie ein `Process`-Objekt für eine Anwendung besitzen (z. B. weil Sie die Methode `IsOtherInstanceRunning` aus dem Rezept 92 aufgerufen haben), können Sie diese ebenfalls in den Vordergrund holen. Dazu lesen Sie den Handle des Hauptfensters über die Eigenschaft `MainWindowHandle` aus und setzen die API-Funktionen des vorhergehenden Rezepts ein. Listing 180 zeigt dies am Beispiel der Methode `ActivateApplicationByProcess`. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Diagnostics`, `System.Text` und `System.Runtime.InteropServices` einbinden.

```
public static void ActivateApplicationByProcess(Process process)
{
   if ((int)process.MainWindowHandle != 0)
   {
      if (IsIconic((IntPtr)process.MainWindowHandle) != 0)
      {
         // Hauptfenster nach vorne holen
         if (OpenIcon(process.MainWindowHandle) == 0)
         {
            throw new Exception("Windows-Fehler " +
               Marshal.GetLastWin32Error() + " beim Wiederherstellen " +
               "des Fensters für den Prozess " + process.ProcessName +
               "' aus dem minimierten Zustand.");
         }
      }

      // Fenster in den Vordergrund holen
      if (SetForegroundWindow(process.MainWindowHandle) == 0)
      {
         throw new Exception("Windows-Fehler " +
            Marshal.GetLastWin32Error() + " beim Nach-vorne-Holen " +
            "des Hauptfensters des Prozesses '" + process.ProcessName +
            "'");
      }
```

Listing 180: Aktivieren einer Anwendung über ein Process-Objekt

```
   }
   else
   {
      throw new Exception("Der Prozess '" + process.ProcessName +
         " besitzt kein Hauptfenster");
   }
}
```

Listing 180: Aktivieren einer Anwendung über ein Process-Objekt (Forts.)

94 Auflisten aller laufenden Prozesse eines Computers

Über die `GetProcesses`-Methode der `Process`-Klasse erhalten Sie ein Array von `Process`-Objekten für alle aktuell ausgeführten Prozesse eines Computers. Wenn Sie dieser Methode nichts übergeben, verwaltet das zurückgegebene Array Objekte für alle lokalen Prozesse. Alternativ können Sie dieser Methode den Namen eines Computers im Netzwerk übergeben, um dessen Prozesse aufzulisten. Die Übergabe einer IP-Adresse oder des Namens *localhost* ist (eigenartigerweise) leider nicht möglich.

Die im Array verwalteten `Process`-Objekte besitzen eine Vielfalt an Eigenschaften und Methoden. Die interessantesten setzt der Beispiel-Programmcode in Listing 181 ein. Das Programm liest die Startzeit und die Prozessorzeit in einem try-Block ein, da für einige Prozesse wie dem Idle-Prozess auf diese Eigenschaften (auch mit Administrator-Rechten) nicht zugegriffen werden kann.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Diagnostics` und `System.ComponentModel` importieren.

```
// Alle auf dem lokalen Computer laufenden Prozesse ermitteln
Process[] processes = Process.GetProcesses();

// Die Prozesse durchgehen und Informationen dazu anzeigen
for (int i = 0; i < processes.Length; i++)
{
   Console.WriteLine("Id: {0}", processes[i].Id);
   Console.WriteLine("Name: {0}", processes[i].ProcessName);
   try
   {
      Console.WriteLine("Startzeit: {0}", processes[i].StartTime.ToString());
   }
   catch (Win32Exception ex)
   {
      Console.WriteLine("Startzeit: Nicht verfügbar: " + ex.Message);
   }
   Console.WriteLine("Priorität: {0}", processes[i].BasePriority);
   Console.WriteLine("Hauptfenster-Titel: {0}", processes[i].MainWindowTitle);
```

Listing 181: Auflisten der wichtigsten Eigenschaften aller Prozesse des lokalen Computers

```
   Console.WriteLine("Prozess reagiert: {0}", processes[i].Responding);
   Console.WriteLine("Anzahl Threads: {0}", processes[i].Threads.Count);
   try
   {
      Console.WriteLine("Prozessor-Zeit: {0}",
         processes[i].TotalProcessorTime);
   }
   catch (Win32Exception ex)
   {
      Console.WriteLine("Prozessor-Zeit: Nicht verfügbar: " + ex.Message);
   }
   Console.WriteLine("Virtuelle Speichergröße: {0}",
      processes[i].VirtualMemorySize64);
   Console.WriteLine();
}
```

Listing 181: Auflisten der wichtigsten Eigenschaften aller Prozesse des lokalen Computers (Forts.)

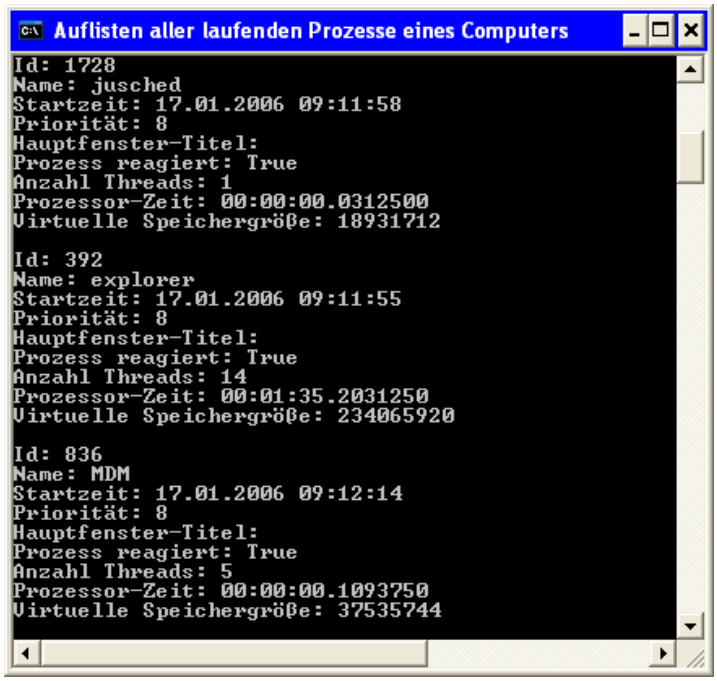


Abbildung 59: Das Programm zum Auflisten aller Prozesse eines Computers in Aktion

Wenn Sie die Prozesse eines Computers im Netzwerk auflisten wollen, geben Sie einfach dessen Namen an:

```
Process[] processes = Process.GetProcesses("Trillian");
...
```

95 Auflisten aller laufenden Anwendungen eines Computers

Das Auflisten aller laufenden Anwendungen ist nicht auf eine einfache Weise möglich. Problematisch daran ist, dass Sie zwar die laufenden Prozesse ermitteln können, aber nicht ohne weiteres erkennen, welcher Prozess eine echte Anwendung und welcher ein Hintergrundprozess ist.

Zum Ermitteln der laufenden Prozesse können Sie wie im vorhergehenden Rezept die `GetProcesses`-Methode der `Process`-Klasse (aus dem Namensraum `System.Diagnostics`) aufrufen, die ein Array von `Process`-Objekten zurückgibt. Die Lösung, Anwendungen von Hintergrundprozessen zu unterscheiden, basiert nun auf der Idee, dass Anwendungen ein Hauptfenster besitzen. Da Sie den Handle dieses Fensters aus der Eigenschaft `MainWindowHandle` auslesen können, ist es ein Leichtes, Anwendungen (mit einer relativ hohen Sicherheit) von Hintergrundprozessen zu unterscheiden.

Das Beispiel erfordert den Import der Namensräume `System` und `System.Diagnostics`.

```
// Alle laufenden Prozesse ermitteln
Process[] processes = Process.GetProcesses();

// Die Prozesse durchgehen und die Prozesse mit einem Hauptfenster ermitteln
for (int i = 0; i < processes.Length; i++)
{
   if ((int)processes[i].MainWindowHandle != 0)
   {
      // Hauptfenster ist vorhanden, der Prozess ist also eine Anwendung:
      // Prozessname und Titel des Hauptfensters auslesen
      Console.WriteLine("Prozessname: {0}\r\nFenstertitel: {1}\r\n",
         processes[i].ProcessName, processes[i].MainWindowTitle);
   }
}
```

Listing 182: Ermitteln aller laufenden Anwendungen

Das Beispiel liest bei den gefundenen Anwendungen den Prozessnamen über die Eigenschaft `ProcessName` und den Titel des Hauptfensters über die Eigenschaft `MainWindowTitle` aus.

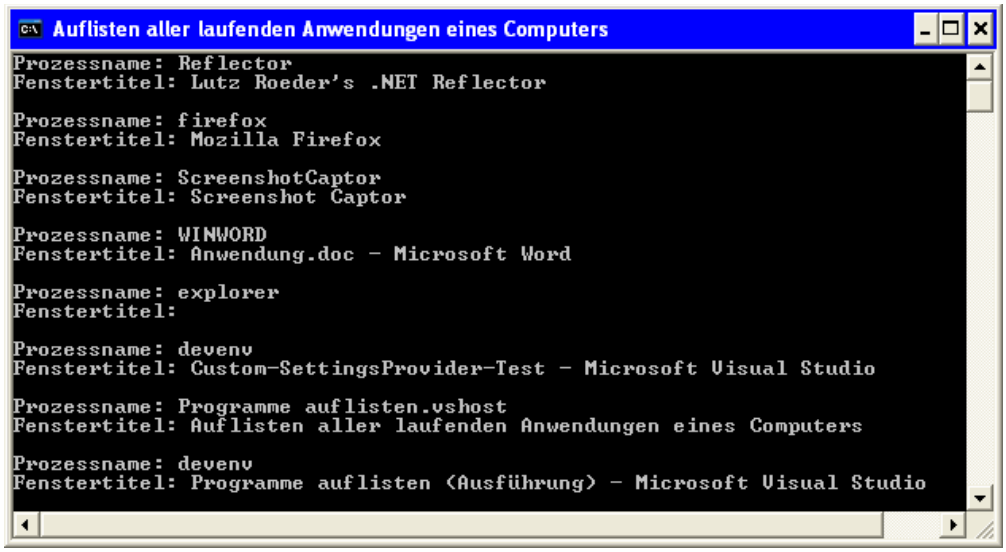


Abbildung 60: Das Programm zum Auflisten aller Anwendungen in Aktion

Hinweis

Der Programmcode ist leider nicht absolut sicher, da einige (wenige) Hintergrundprozesse ein (unsichtbares) Hauptfenster besitzen und diese dann auch mit aufgelistet werden.

96 Informationen über die Dienste eines Computers auslesen

Über die Methode `GetServices` der Klasse `System.ServiceProcess.ServiceController` erhalten Sie ein Array von Referenzen auf `ServiceController`-Instanzen, die die Dienste eines Computers repräsentieren. Rufen Sie `GetServices` ohne Argument oder mit einem Punkt als Computernamen auf, erhalten Sie die Dienste des lokalen Computers. Sie können aber auch den Namen eines entfernten Rechners angeben, um dessen Dienste auszulesen.

Die `ServiceController`-Objekte der einzelnen Dienste liefern eine Vielzahl an Informationen und bieten darüber hinaus sogar Möglichkeiten, Dienste zu starten, anzuhalten oder zu stoppen (siehe nächstes Rezept). Der folgende Quellcode listet die wichtigsten Daten der ermittelten Dienste auf.

Das Programm benötigt eine Referenz auf die Assemblys *System.ServiceProcess.dll* und *System.Windows.Forms.dll.* und die Einbindung der Namensräume `System`, `System.ServiceProcess` und `System.Windows.Forms`. Unter Windows 98 und Me kann das Programm nicht ausgeführt werden, da diese Systeme keine Dienste unterstützen.

```
string machineName = "."; // . = lokaler Rechner

// Dienste ermitteln
ServiceController[] services = null;
try
{
   services = ServiceController.GetServices(machineName);
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Auslesen der Dienste " + "auf dem Computer " +
machineName + ": " + ex.Message, Application.ProductName, MessageBoxButtons.OK,
MessageBoxIcon.Error);
   return ;
}

// Alle Dienste durchgehen und die wichtigsten
// Informationen ausgeben
for (int i = 0; i < services.Length; i++)
{
   Console.WriteLine();

   // Dienstname
   Console.WriteLine("Dienstname: {0}", services[i].ServiceName);

   // Anzeigename
   Console.WriteLine("Anzeigename: {0}", services[i].DisplayName);

   // Status
   Console.Write("Status: ");
   switch (services[i].Status)
   {
      case ServiceControllerStatus.Running:
         Console.WriteLine("Wird ausgeführt");
         break;

      case ServiceControllerStatus.Stopped:
         Console.WriteLine("Gestoppt");
         break;

      case ServiceControllerStatus.Paused:
         Console.WriteLine("Angehalten");
         break;

      case ServiceControllerStatus.StartPending:
         Console.WriteLine("Wird gestartet");
```

Listing 183: Auflisten von Informationen über die Dienste eines Computers

```
            break;

        case ServiceControllerStatus.ContinuePending:
            Console.WriteLine("Wird wieder gestartet");
            break;

        case ServiceControllerStatus.PausePending:
            Console.WriteLine("Wird angehalten");
            break;

        case ServiceControllerStatus.StopPending:
            Console.WriteLine("Wird beendet");
            break;
    }

    // Dienst-Typ
    Console.Write("Dienst-Typ: ");
    string serviceType = null;
    if ((services[i].ServiceType & ServiceType.Adapter) > 0)
    {
        if (serviceType != null)
        {
            serviceType += ", ";
        }
        serviceType += "Adapter";
    }

    if ((services[i].ServiceType & ServiceType.FileSystemDriver) > 0)
    {
        if (serviceType != null)
        {
            serviceType += ", ";
        }
        serviceType += "Dateisystem-Treiber";
    }

    if ((services[i].ServiceType & ServiceType.InteractiveProcess) > 0)
    {
        if (serviceType != null)
        {
            serviceType += ", ";
        }
        serviceType += "Interaktiver Prozess";
    }
```

Listing 183: Auflisten von Informationen über die Dienste eines Computers (Forts.)

```
   if ((services[i].ServiceType & ServiceType.KernelDriver) > 0)
   {
      if (serviceType != null)
      {
         serviceType += ", ";
      }
      serviceType += "Kernel-Treiber";
   }

   if ((services[i].ServiceType & ServiceType.RecognizerDriver) > 0)
   {
      if (serviceType != null)
      {
         serviceType += ", ";
      }
      serviceType += "Dateisystem-Ermittlungs-Treiber";
   }

   if ((services[i].ServiceType & ServiceType.Win32OwnProcess) > 0)
   {
      if (serviceType != null)
      {
         serviceType += ", ";
      }
      serviceType += "Win32-Anwendung in eigenem Prozess";
   }

   if ((services[i].ServiceType & ServiceType.Win32ShareProcess) > 0)
   {
      if (serviceType != null)
      {
         serviceType += ", ";
      }
      serviceType += "Win32-Anwendung in " + "gemeinsamen Prozess";
   }
   Console.WriteLine(serviceType);
}
```

Listing 183: Auflisten von Informationen über die Dienste eines Computers (Forts.)

Neben den im Beispielprogramm dargestellten Informationen können Sie noch weitere Informationen abrufen, wie z. B., ob ein Dienst beendet oder angehalten werden kann (`CanPauseAndContinue`, `CanShutdown`, `CanStop`) oder von welchen anderen Diensten ein Dienst abhängig ist (`ServicesDependedOn`).

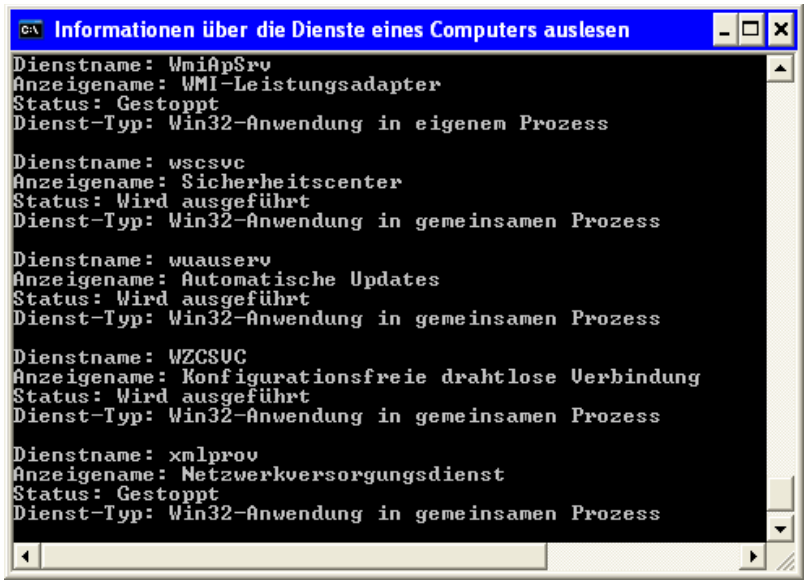


Abbildung 61: Auflisten von Informationen über die Dienste eines Computers

97 Dienste starten, anhalten, fortfahren und stoppen

Eine Instanz der Klasse `ServiceController` ermöglicht neben dem Auslesen von Informationen zu Diensten (siehe Rezept 96) auch, diese zu starten, anzuhalten, fortzufahren oder zu stoppen. Dazu verwenden Sie die Methoden `GetServiceStatus`, `IsServiceRunning` `Start`, `Pause`, `Continue` und `Stop`.

Beim Erzeugen des `ServiceController`-Objekts geben Sie den Dienstnamen an. Im optionalen zweiten Argument können Sie den Namen des Computers angeben, auf dem der Dienst bearbeitet werden soll.

Beim Starten, Pausieren, Fortfahren und Stoppen müssen Sie einiges beachten. So kann ein Dienst nur erfolgreich gestartet werden, wenn er den Status `Stopped` besitzt. Sogar bei einem Dienst, der bereits ausgeführt wird, erzeugt die `Start`-Methode eine Ausnahme, was eigentlich unsinnig ist. Eine tolerantere Methode sollte den Status abfragen und in dem Fall, dass der Dienst bereits läuft oder gerade gestartet wird, einfach nichts machen. Ähnlich verhält es sich beim Stoppen, Pausieren oder Fortfahren eines Dienstes.

Daneben kann es einige Zeit dauern, bis ein Dienst gestartet, gestoppt wurde, pausiert oder fortgefahren ist. Die entsprechenden Methoden kehren aber sofort zurück, ohne auf die erfolgreiche Ausführung zu warten. Wenn Sie sich nach dem Aufruf der Methode darauf verlassen müssen, dass der Dienst seinen Status wie gewünscht geändert hat, müssen Sie über die Methode `WaitForStatus`, der Sie einen Dienst-Status und optional einen Timeout (in Form eines `TimeSpan`-Objekts) übergeben, darauf warten, dass der Dienst den gewünschten Status annimmt.

Die folgende Klasse `ServiceUtil` setzt dies um und stellt einige Methoden zur Arbeit mit Diensten zur Verfügung. Zum Kompilieren müssen Sie eine Referenz auf die Assembly *System.ServiceProcess.dll* anlegen und die Namensräume `System` und `System.ServiceProcess` ein-

binden. Ein Programm, das diese Klasse einsetzt, können Sie nicht unter Windows 98 und Me ausführen, da Dienste erst ab Windows NT unterstützt werden.

Die Methoden `GetServiceStatus` und `IsServiceRunning` liefern Informationen zu einem Dienst:


```
public static class ServiceUtil
{
   /* Liefert den Status eines Dienstes */
   public static ServiceControllerStatus GetServiceStatus(string serviceName,
      string machineName)
   {
      // Dienst ermitteln
      if (machineName == null)
      {
         machineName = ".";
      }
      ServiceController service = new ServiceController(
         serviceName, machineName);

      // Status zurückgeben
      return service.Status;
   }

   /* Überprüft, ob ein Dienst ausgeführt wird (oder gerade startet) */
   public static bool IsServiceRunning(string serviceName, string machineName)
   {
      // Status ermitteln
      ServiceControllerStatus status =
         ServiceUtil.GetServiceStatus(serviceName, machineName);

      // Status auswerten
      return (status == ServiceControllerStatus.Running ||
         status == ServiceControllerStatus.StartPending ||
         status == ServiceControllerStatus.ContinuePending);
   }
```

Listing 184: Methoden zum Ermitteln von Informationen zu einem Dienst

Die Methode `StartService` startet einen Dienst auf einem anzugebenden Computer (`null` oder "." für den lokalen Rechner). Am dritten Argument kann ein String-Array übergeben werden, dessen Strings als Startargumente für den Dienst dienen. Die Methode akzeptiert aber auch `null`, wenn keine Argumente übergeben werden sollen. Das letzte Argument gibt einen Timeout für den Start an. Nach dem Starten des Dienstes wartet `StartService` bis maximal zu diesem Timeout darauf, dass der Dienst den Status `Running` annimmt. `StartService` liefert den Status des Dienstes zurück, damit dieser nach dem Aufruf überprüft werden kann. Den Rest erläutern die Kommentare:

```
public static ServiceControllerStatus StartService(string serviceName,
   string machineName, string[] arguments, TimeSpan timeout)
{
   // Dienst ermitteln
   if (machineName == null)
   {
      machineName = ".";
   }
   ServiceController service = new ServiceController(
      serviceName, machineName);

   // Dienst starten, falls dieser nicht bereits gestartet ist oder wird
   if (service.Status != ServiceControllerStatus.Running &&
      service.Status != ServiceControllerStatus.StartPending &&
      service.Status != ServiceControllerStatus.ContinuePending)
   {
      if (arguments != null)
      {
         service.Start(arguments);
      }
      else
      {
         service.Start();
      }
   }

   // Den übergebenen Timeout warten,
   // bis der Dienst den Status Running besitzt
   service.WaitForStatus(ServiceControllerStatus.Running, timeout);

   // Status des Dienstes zurückliefern
   return service.Status;
}
```

Listing 185: Methode zum toleranten Starten eines Dienstes

Die Methode `PauseService` besitzt ähnliche Argumente wie `StartService` (lediglich ohne die Dienst-Start-Argumente) und hält einen Dienst an. `PauseService` fragt zusätzlich zu dem aktuellen Status des Dienstes über die Eigenschaft `CanPauseAndContinue` ab, ob dieser angehalten werden kann, und erzeugt im negativen Fall eine Ausnahme. Nach dem Aufruf der `Pause`-Methode wartet `PauseService` maximal bis zum Ablauf des Timeout darauf, dass der Dienst den Status `Paused` annimmt. Zur Überprüfung liefert diese Methode wieder den Status des Dienstes zurück:

```
public static ServiceControllerStatus PauseService(string serviceName,
   string machineName, TimeSpan timeout)
{
```

Listing 186: Methode zum toleranten Anhalten eines Dienstes

```
    // Dienst ermitteln
    if (machineName == null)
    {
       machineName = ".";
    }
    ServiceController service = new ServiceController(
       serviceName, machineName);

    // Überprüfen, ob der Dienst angehalten werden kann
    if (service.CanPauseAndContinue == false)
    {
       throw new Exception("Der Dienst '" + serviceName +
          "' kann nicht angehalten werden");
    }

    // Dienst anhalten, falls dieser nicht bereits angehalten ist
    if (service.Status != ServiceControllerStatus.Paused &&
       service.Status != ServiceControllerStatus.PausePending &&
       service.Status != ServiceControllerStatus.StopPending)
    {
       service.Pause();
    }

    // Den übergebenen Timeout warten,
    // bis der Dienst den Status Paused besitzt
    service.WaitForStatus(ServiceControllerStatus.Paused, timeout);

    // Status des Dienstes zurückliefern
    return service.Status;
 }
```

Listing 186: Methode zum toleranten Anhalten eines Dienstes (Forts.)

Die Methode `ContinueService`, die der Methode `StartService` ähnlich ist, fährt einen angehaltenen Dienst fort:

```
public static ServiceControllerStatus ContinueService(string serviceName,
    string machineName, TimeSpan timeout)
 {
    // Dienst ermitteln
    if (machineName == null)
    {
       machineName = ".";
    }
    ServiceController service = new ServiceController(
       serviceName, machineName);
```

Listing 187: Methode zum Fortfahren eines Dienstes

```
    // Überprüfen, ob der Dienst fortgefahren werden kann
    if (service.CanPauseAndContinue == false)
    {
       throw new Exception("Der Dienst '" + serviceName +
          "' kann nicht fortgefahren werden");
    }

    // Dienst fortfahren, falls dieser nicht bereits ausgeführt wird
    // oder gerade startet
    if (service.Status != ServiceControllerStatus.Running &&
       service.Status != ServiceControllerStatus.StartPending &&
       service.Status != ServiceControllerStatus.ContinuePending)
    {
       service.Continue();
    }

    // Den übergebenen Timeout warten, bis der Dienst den
    // Status Running besitzt
    service.WaitForStatus(ServiceControllerStatus.Running, timeout);

    // Status des Dienstes zurückliefern
    return service.Status;
 }
```

Listing 187: Methode zum Fortfahren eines Dienstes (Forts.)

Nun fehlt nur noch eine Methode zum Stoppen von Diensten:

```
public static ServiceControllerStatus StopService(string serviceName,
    string machineName, TimeSpan timeout)
 {
    // Dienst ermitteln
    if (machineName == null)
    {
       machineName = ".";
    }
    ServiceController service = new ServiceController(
       serviceName, machineName);

    // Überprüfen, ob der Dienst gestoppt werden kann
    if (service.CanStop == false)
    {
       throw new Exception("Der Dienst '" + serviceName +
          "' kann nicht gestoppt werden");
    }
```

Listing 188: Methode zum Stoppen eines Dienstes

```
      // Dienst stoppen, falls dieser nicht gestoppt ist oder gerade stoppt
      if (service.Status != ServiceControllerStatus.Stopped &&
         service.Status != ServiceControllerStatus.StopPending)
      {
         service.Stop();
      }

      // Warten, bis der Dienst den Status Stopped besitzt
      service.WaitForStatus(ServiceControllerStatus.Stopped, timeout);

      // Status des Dienstes zurückliefern
      return service.Status;
   }
}
```

Listing 188: Methode zum Stoppen eines Dienstes (Forts.)

Alle Methoden überprüfen zwar über den Status und die Eigenschaften `CanPauseAndContinue` bzw. `CanStop`, ob der gewünschte Statuswechsel grundsätzlich möglich bzw. nötig ist, fragen aber beim Aufruf der Methode, die den Statuswechsel bewirkt, nicht mehr weiter nach, ob dieser möglich ist. Ausnahmen, die im Fehlerfall generiert werden, werden auf diese Weise einfach an den Aufrufer weitergegeben. Die `Stop`-Methode führt z. B. nur dann zum Erfolg, wenn der Dienst gerade ausgeführt wird (also den Status `Running` besitzt). Da die im Fehlerfall erzeugten Ausnahmen recht aussagekräftig sind, wollte ich den Programmcode nicht noch über weitere Überprüfungen bzw. Ausnahmebehandlungen unübersichtlicher machen.

Das folgende Listing zeigt eine beispielhafte Anwendung der `StartService`-Methode. Gestartet wird der Dienst *MSSQLSERVER* auf dem lokalen Computer ohne Argumente. Als Timeout werden drei Sekunden übergeben:

```
ServiceControllerStatus serviceStatus;
string serviceName = "MSSQLSERVER";
try
{
   Console.WriteLine("Starte " + serviceName + "...");
   serviceStatus = ServiceUtil.StartService(serviceName, null, null,
      new TimeSpan(0, 0, 0, 3));
   if (serviceStatus != ServiceControllerStatus.Running)
   {
      Console.WriteLine("Der Dienst konnte nicht innerhalb " +
         "des Timeout gestartet werden");
      return;
   }
}
catch (Exception ex)
```

Listing 189: Beispiel-Anwendung der StartService-Methode

```
{
   Console.WriteLine("Fehler beim Starten: {0}", ex.Message);
   Console.ReadLine();
   return;
}
```

Listing 189: Beispiel-Anwendung der StartService-Methode (Forts.)

98 Ermitteln, ob eine Anwendung ausgeführt wird

Wenn Sie in einem Ihrer Programme überprüfen müssen, ob eine andere Anwendung bereits ausgeführt wird (um diese im negativen Fall gegebenenfalls zu starten, wie ich es im Rezept 99 beschreibe), können Sie dazu prinzipiell die `Process`-Klasse verwenden. Dazu erzeugen Sie über die `GetProcessesByName`-Methode ein Array der Prozesse, die den Namen der Anwendung tragen. Enthält das alle Elemente, können Sie davon ausgehen, dass die Anwendung bereits ausgeführt wird.

Problematisch ist dabei allerdings, dass die Prozessnamen verschiedener Anwendungen identisch sein können. Deshalb können Sie nicht mit Sicherheit sagen, dass genau die Anwendung, die Sie meinen, bereits ausgeführt wird.

Zur Lösung dieses Problems können Sie den Titel des Hauptfensters der Anwendung überprüfen, wie ich es in dem oben genannten Rezept programmiert habe. Diese Titel werden bei verschiedenen Anwendungen dynamisch zusammengesetzt (bei vielen Anwendungen wird z. B. der Name einer geladenen Datei mit im Titel dargestellt). Deshalb sollten Sie lediglich überprüfen, ob ein für die Anwendung typischer Begriff im Titel vorkommt.

Die Methode `IsApplicationRunning` in Listing 190 nimmt Ihnen diese Arbeit ab. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```
public static bool IsApplicationRunning(string name, string title)
{
   // Ermitteln der Prozesse, die den übergebenen Namen tragen
   Process[] processes = Process.GetProcessesByName(name);

   // Überprüfen, ob einer der eventuell gefundenen Prozesse im Titel
   // des Hauptfensters den übergebenen Titel-String besitzt
   foreach (Process process in processes)
   {
      if (process.MainWindowTitle.ToLower().IndexOf(title.ToLower()) > 0)
      {
         return true;
      }
   }
```

Listing 190: Methode zur Überprüfung, ob eine bestimmte Anwendung ausgeführt wird

```
   // Wenn die Methode hier ankommt, wurde kein Prozess gefunden
   return false;
}
```

Listing 190: Methode zur Überprüfung, ob eine bestimmte Anwendung ausgeführt wird (Forts.)

Im folgenden Listing finden Sie eine beispielhafte Anwendung dieser Methode. Überprüft wird, ob Microsoft Outlook bereits ausgeführt wird.

```
// Überprüfen, ob Outlook läuft
if (IsApplicationRunning("Outlook", "Microsoft Outlook"))
{
   Console.WriteLine("Outlook läuft");
}
else
{
   Console.WriteLine("Outlook läuft nicht");
}
```

Listing 191: Beispielhafte Anwendung der Methode zur Überprüfung, ob eine Anwendung bereits ausgeführt wird

99 Anwendungen starten

Anwendungen können Sie über die `Start`-Methode der `Process`-Klasse aus dem Namensraum `System.Diagnostics` starten. Dazu können Sie am ersten Argument den Dateinamen und im optionalen zweiten Argument Befehlszeilenargumente für den Start übergeben. Flexibler ist allerdings die Übergabe eines `ProcessStartInfo`-Objekts, das Sie mit genaueren Angaben für den Start initialisieren können. So können Sie z. B. über die Eigenschaft `WindowStyle` festlegen, dass das Anwendungsfenster maximiert geöffnet wird.

Das Programm in Listing 192 startet den Windows-Editor maximiert und übergibt als Befehlszeilenargument die Datei *eula.txt*, die im Windows-Systemverzeichnis gespeichert ist. Ausnahmen, die beim Start der Anwendung erzeugt werden können (z. B. weil die Anwendung nicht vorhanden ist), werden abgefangen. Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.IO` und `System.Diagnostics` einbinden.

```
// Dateinamen zusammensetzen
string exeFileName = Path.Combine(Environment.SystemDirectory, "notepad.exe");
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Anwendungs-Prozess mit einem maximierten Fenster starten
ProcessStartInfo psi = new ProcessStartInfo(exeFileName, arguments);
psi.WindowStyle = ProcessWindowStyle.Maximized;
```

Listing 192: Starten einer Anwendung

```
try
{
   Process.Start(psi);
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Starten der Anwendung: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Listing 192: Starten einer Anwendung (Forts.)

Ist das zu startende Programm in einem Ordner gespeichert, der in der Umgebungsvariable `Path` eingetragen ist, muss der Pfad zum Ordner nicht angegeben werden. In diesem Fall können Sie aber nicht sicher sein, dass das gewünschte Programm gestartet wird. Schließlich kann ein Programm mit demselben Dateinamen auch in einem anderen Ordner gespeichert sein.

100 Dateien mit der assoziierten Anwendung öffnen

In Windows sind viele Dateitypen mit einer Anwendung assoziiert. Wenn Sie eine Datei mit der verknüpften Anwendung öffnen wollen, können Sie dazu die `Start`-Methode der Klasse `System.Diagnostics.Process` verwenden. Am ersten Argument übergeben Sie wie im vorhergehenden Rezept ein `ProcessStartInfo`-Objekt mit Informationen zum Start. Neben dem Dateinamen, den Sie bei der Erzeugung dieses Objekts im Konstruktor übergeben können, legen Sie dann über die Eigenschaft `UseShellExecute` fest, dass die `Start`-Methode beim Starten des Prozesses die Shell verwendet (so, als würden Sie die Datei im Explorer öffnen). Über die Eigenschaft `WindowStyle` können Sie festlegen, wie das Fenster geöffnet werden soll (normal, minimiert, maximiert).

Die Methode `OpenFileWithAssociatedApp` in Listing 193 setzt dies um. Der am ersten Argument übergebene Dateiname wird dem Konstruktor des erzeugten `ProcessStartInfo`-Objekts übergeben. Wenn am zweiten Argument `true` übergeben wird, dass das Anwendungsfenster maximiert geöffnet werden soll, setzt `OpenFileWithAssociatedApp` die Eigenschaft `WindowStyle` auf `ProcessWindowStyle.Maximized`. Ausnahmen werden nicht abgefangen, sodass diese an den Aufrufer weitergegeben werden. Die Methode erfordert das Einbinden der Namensräume `System` und `System.Diagnostics`.

```
public static void OpenFileWithAssociatedApp(string fileName, bool maximized)
{
   // ProcessStartInfo-Instanz erzeugen und initialisieren
   ProcessStartInfo psi = new ProcessStartInfo(fileName);
   if (maximized)
   {
      psi.WindowStyle = ProcessWindowStyle.Maximized;
   }
```

Listing 193: Methode zum Öffnen einer Datei mit der assoziierten Anwendung

```
   psi.UseShellExecute = true;

   // Prozess starten
   Process.Start(psi);
}
```

Listing 193: Methode zum Öffnen einer Datei mit der assoziierten Anwendung (Forts.)

101 Starten einer anderen Anwendung und warten, bis diese beendet ist

In einigen Fällen muss ein Programm eine andere Anwendung starten und mit der weiteren Ausführung warten, bis die andere Anwendung beendet wurde. Die Lösung dieses Problems zeigt die Methode `StartApplication`, die eine Anwendung über die `Start`-Methode der `Process`-Klasse (siehe Rezept 99) startet und abhängig vom Argument `waitForExit` so lange wartet, bis diese beendet ist. `StartApplication` ruft dazu die Methode `WaitForExit` auf, die so auf das Beenden des gestarteten Prozesses wartet, dass Windows dabei nicht ausgebremst wird. Zum Kompilieren müssen Sie die Namensräume `System` und `System.Diagnostics` importieren.

```
public static void StartApplication(string path, string arguments,
   bool maximized, bool waitForExit)
{
   // ProcessStartInfo-Instanz erzeugen und initialisieren
   ProcessStartInfo psi = new ProcessStartInfo(path);
   if (arguments != "")
   {
      psi.Arguments = arguments;
   }
   if (maximized)
   {
      psi.WindowStyle = ProcessWindowStyle.Maximized;
   }
   psi.UseShellExecute = true;

   // Prozess starten
   Process process = Process.Start(psi);

   // Auf die Beendigung des Prozesses warten, sofern dies gewünscht ist
   if (waitForExit)
   {
      process.WaitForExit();
   }
}
```

Listing 194: Methode zum Starten einer Anwendung mit optionalem Warten auf deren Beendigung

Das folgende Listing zeigt die Anwendung dieser Methode:

```
// Dateinamen zusammensetzen
string exeFileName = Path.Combine(Environment.SystemDirectory, "notepad.exe");
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Anwendung starten und warten, bis diese beendet ist
Console.WriteLine("Starte '{0}' und warte auf die Beendigung ... ",
   exeFileName);
StartApplication(exeFileName, arguments, false, true);
Console.WriteLine("Anwendung wurde beendet");
```

Listing 195: Anwendung der Methode zum Starten einer Anwendung mit Warten auf deren Beendigung

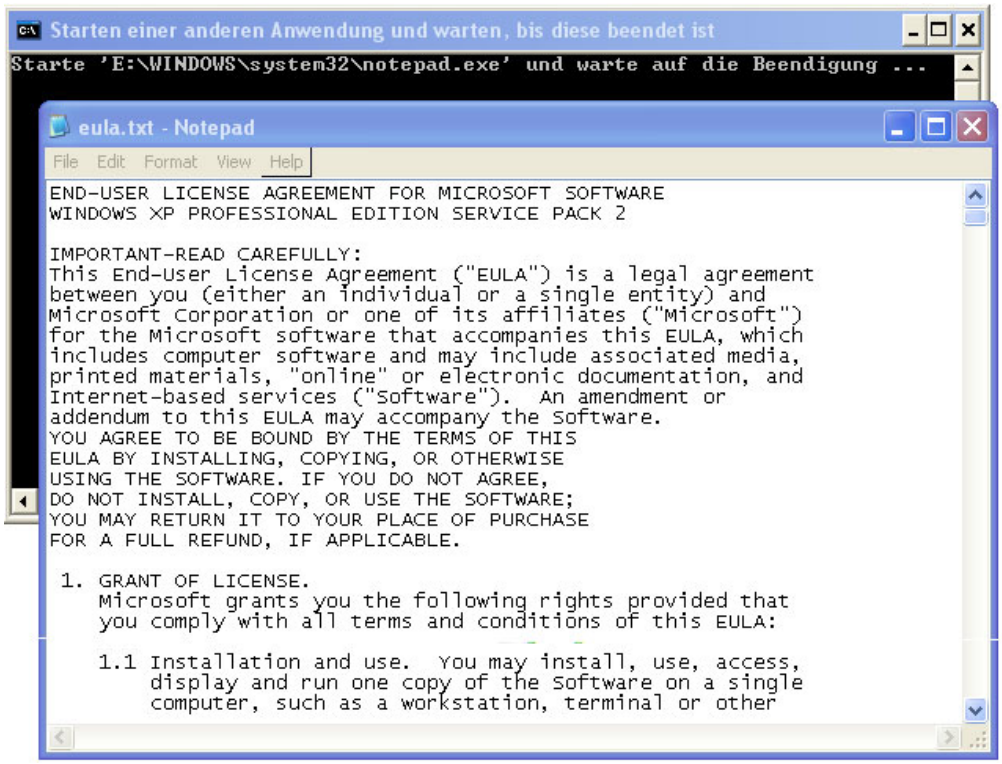


Abbildung 62: Das Beispielprogramm hat den Editor gestartet und wartet auf dessen Beendigung

102 Konsolenanwendungen starten und die Ausgabe auswerten

Auch im Windows-Zeitalter gibt es noch eine Menge mehr oder weniger hilfreiche Konsolenanwendungen. Das für C#-Entwickler beste Beispiel ist der C#-Compiler *csc.exe*. Der Aufruf solcher Anwendungen in einem Programm ist kein Problem und wurde bereits in den vorherigen Rezepten besprochen. In einigen Fällen werden Sie jedoch die Ausgabe der aufgerufenen Konsolenanwendung auswerten wollen. Und das beschreibe ich in diesem Rezept.

Zunächst benötigen Sie ein `ProcessStartInfo`-Objekt (aus dem Namensraum `System.Diagnostics`), das Sie im Konstruktor mit dem Dateinamen der zu startenden Anwendung initialisieren. Das Beispiel in Listing 196 erzeugt ein solches Objekt für das Windows-Zubehör-Programm *ipconfig* (das die aktuelle IP-Konfiguration an der Konsole ausgibt). Wenn Sie Argumente übergeben wollen (oder müssen), schreiben Sie diese in die Eigenschaft `Arguments`.

Eine Konsolenanwendung schreibt normale Informationen in den Standard-Ausgabe- und Fehlerinformationen (normalerweise) in den Standard-Fehler-Kanal. Diese Kanäle geben ihre Informationen per Voreinstellung an der Konsole aus. Da Sie die Ausgabe des Programms umleiten wollen, setzen Sie die Eigenschaften `RedirectStandardOutput` und `RedirectStandardError` auf `true`. `UseShellExecute` müssen Sie in diesem Fall auf `false` setzen, da die Umleitung ansonsten nicht möglich ist. `UseShellExecute` legt fest, ob zum Starten des Prozesses die Betriebssystemshell verwendet wird. Ist diese Eigenschaft `false`, wird der Prozess direkt über die ausführbare Datei gestartet.

Dann starten Sie den Prozess über die `Start`-Methode der `Process`-Klasse, der Sie die `ProcessStartInfo`-Instanz übergeben. Über die `WaitForExit`-Methode warten Sie auf das Ende des Prozesses. Hier sollten Sie am ersten Argument einen Timeout in Millisekunden übergeben, da es vorkommen kann, dass die Ausführung der Konsolenanwendung blockiert wird (z. B. weil diese den Anwender fragt, ob eine bestimmte Aktion wirklich ausgeführt werden soll).

Die `Start`-Methode gibt eine Referenz auf das `Process`-Objekt zurück, das den gestarteten Prozess repräsentiert. Sie sollten diese Referenz in eine Variable schreiben.

Um auszuwerten, ob der Prozess innerhalb des Timeout beendet wurde, fragen Sie die `HasExited`-Eigenschaft des `Process`-Objekts ab. Ist diese `true`, sollten Sie zunächst ermitteln, ob ein Fehler aufgetreten ist. Viele Konsolenanwendungen geben dazu einen Integerwert zurück, den Sie über die Eigenschaft `ExitCode` erreichen. Der Wert 0 bedeutet normalerweise, dass kein Fehler aufgetreten ist. Verlassen können Sie sich darauf aber nicht, eine Anwendung ist nicht gezwungen, den Rückgabewert zu setzen. Sie müssen die Rückgabe des Programms im Einzelfall also immer prüfen.

Über die Eigenschaft `StandardError` können Sie den Fehlerkanal auslesen. Die eigentliche Ausgabe des Programms erreichen Sie über `StandardOutput`. Diese Eigenschaften referenzieren je einen `StreamReader`.

Das folgende Listing zeigt am Beispiel von *ipconfig*, wie Sie eine »normale« Konsolenanwendung starten und deren Ergebnis auswerten. Das Beispiel geht davon aus, dass die Anwendung bei einem Fehler einen Rückgabewert ungleich 0 zurückgibt und die Fehlermeldung in den Fehlerkanal schreibt.

Zum Kompilieren des Beispiels müssen Sie den Namensraum `System.Diagnostics` importieren.

```
// ProcessStartInfo-Objekt erzeugen und initialisieren
ProcessStartInfo psi = new ProcessStartInfo("ipconfig");
psi.Arguments = "/all";
psi.UseShellExecute = false;
psi.RedirectStandardOutput = true;
psi.RedirectStandardError = true;

// Prozess starten und auf dessen Ende warten
Process process = Process.Start(psi);
int timeout = 3000;
process.WaitForExit(timeout);

// Das Ergebnis auswerten
string errors = null;
string result = null;
if (process.HasExited)
{
   // Der Prozess wurde innerhalb des Timeout beendet:
   // Fehler auslesen
   if (process.ExitCode != 0)
   {
      errors = "Der Prozess wurde mit dem Code " +
         process.ExitCode + " beendet.";
   }
   string standardError = process.StandardError.ReadToEnd();
   if (string.IsNullOrEmpty(standardError) == false)
   {
      if (errors != null)
      {
         errors += Environment.NewLine;
      }
      errors += standardError;
   }

   // Das Ergebnis auslesen
   result = process.StandardOutput.ReadToEnd();
}
else
{
   // Der Prozess wurde innerhalb des Timeout nicht beendet
   errors = "Der Prozess konnte innerhalb des Timeout von " +
      timeout + " Millisekunden nicht beendet werden";
}
```

Listing 196: Starten einer Konsolenanwendung (ipconfig) und Auswerten der Ausgabe dieses Programms

```
// Das Ergebnis auswerten
if (errors == null)
{
   // Kein Fehler aufgetreten
   ...
}
else
{
   // Fehler aufgetreten
   ...
}
```

Listing 196: Starten einer Konsolenanwendung (ipconfig) und Auswerten der Ausgabe dieses Programms (Forts.)

Abbildung 63 zeigt eine Anwendung, die auf diese Weise *ipconfig* mit dem Argument */all* aufgerufen und das Ergebnis in eine `TextBox` geschrieben hat.

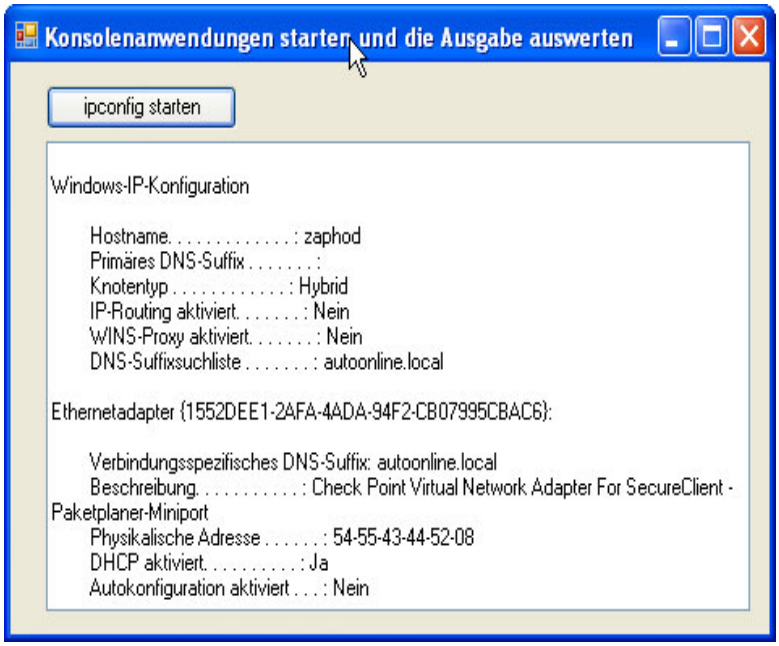


Abbildung 63: Eine Beispielanwendung hat ipconfig /all aufgerufen und die Ausgabe in eine TextBox geschrieben

103 Andere Anwendungen beenden

Wenn Sie eine andere Anwendung beenden wollen, können Sie dies über die `Process`-Klasse erreichen. Um die laufenden Prozesse einer Anwendung zu lokalisieren, rufen Sie die Methode `GetProcessesByName` auf. Diese Methode gibt ein Array von Referenzen auf `Process`-Objekte zurück. Über deren Methode `CloseMainWindow` können Sie dem Hauptfenster die Windows-Nachricht zum Schließen (und damit zum Beenden der Anwendung) senden. Das Schließen

funktioniert natürlich (wie üblich) nur dann, wenn kein modaler Dialog aktiv ist und wenn die Anwendung nicht nachfragt, ob geänderte Daten gespeichert werden sollen (o. Ä.). Statt `CloseMainWindow` können Sie auch `Kill` aufrufen, um den Prozess abzuschießen (ähnlich wie mit dem Taskmanager).

Die Methode `CloseApplication` in Listing 197 setzt diese Technik ein. Um das Schließen ein wenig sicherer zu machen, erwartet diese Methode neben dem Prozessnamen den Titel des Hauptfensters (an dem Sie allerdings auch null übergeben können, wenn Sie alle Prozesse mit dem übergebenen Namen beenden wollen). Das dritte Argument (`waitForExitTimeout`) bestimmt einen Timeout zum Warten auf das Beenden der Anwendung in Millisekunden.

`CloseApplication` ermittelt zunächst alle Prozesse mit dem übergebenen Namen und geht diese durch. Für jeden Prozess überprüft die Methode, ob dessen Hauptfenster den eventuell übergebenen Titel trägt, und ruft dann die `CloseMainWindow`-Methode auf. Wenn am Argument `waitForExitTimeout` ein Wert größer 0 übergeben wurde, ruft `CloseApplication` danach die `WaitForExit`-Methode des `Process`-Objekts unter Übergabe des Timeout auf um auf das Beenden zu warten. Dieser Timeout ist sinnvoll für Anwendungen, die beim Beenden eventuell nachfragen, ob geänderte Daten gespeichert werden sollen.

`CloseApplication` erzeugt eine Instanz der Struktur `ProcessCloseInfo`, in der zunächst in dem Feld `ProcessCount` gespeichert wird, wie viele Prozesse geschlossen werden sollen. Konnte ein Prozess beendet werden, wird das Feld `ClosedProcesses` inkrementiert. Beim Beenden ohne Timeout wartet `CloseApplication` zur Sicherheit 100 Millisekunden um dem Prozess Zeit zu geben, sich zu beenden, und fragt dann die Eigenschaft `HasExited` ab. Ist diese Eigenschaft `true`, wird `ClosedProcesses` hochgezählt. Die `ProcessCloseInfo`- Instanz wird schließlich zurückgegeben.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` importieren.

```
/* Speichert Informationen zu den zu schließenden und den geschlossenen
   Prozessen */
public struct ProcessCloseInfo
{
   /* Gibt die Anzahl der Prozesse an, die zum Schließen ermittelt wurden */
   public int ProcessCount;

   /* Gibt die Anzahl der Prozesse an, die im angegebenen Timeout
      geschlossen werden konnten */
   public int ClosedProcesses;
}

/* Schließt eine bestimmte Anwendung */
public static ProcessCloseInfo CloseApplication(string processName,
   string mainWindowTitle, int waitForExitTimeout)
{
```

Listing 197: Methode zum Beenden von Prozessen

```
   ProcessCloseInfo result = new ProcessCloseInfo();

   // Liste der Prozesse holen, die den übergebenen Prozessnamen tragen
   Process[] processes = Process.GetProcessesByName(processName);
   result.ProcessCount = processes.Length;
   foreach (Process process in processes)
   {
      // Überprüfen des Hauptfenster-Titels, sofern dieser übergeben wurde
      if ((mainWindowTitle != null &&
         process.MainWindowTitle == mainWindowTitle) ||
         (mainWindowTitle == null))
      {
         // Eine Beenden-Nachricht an das Hauptfenster senden
         process.CloseMainWindow();

         // Auf das Beenden des Prozesses warten, wenn ein Timeout
         // übergeben wurde
         if (waitForExitTimeout > 0)
         {
            if (process.WaitForExit(waitForExitTimeout))
            {
               result.ClosedProcesses++;
            }
         }
         else
         {
            // Etwas warten, um dem Prozess Zeit zu lassen sich zu beenden
            Thread.Sleep(100);
            if (process.HasExited)
            {
               result.ClosedProcesses++;
            }
         }
      }
   }

   // Anzahl der beendeten Prozesse zurückgeben
   return result;
}
```

Listing 197: Methode zum Beenden von Prozessen (Forts.)

104 Java-Anwendungen starten

Auch im .NET-Zeitalter (oder vielleicht gerade deswegen) besitzt Java Bedeutung. Und möglicherweise wird es sogar vorkommen, dass eines Ihrer .NET-Programme eine Anwendung starten muss, die mit dem Klassenfeind programmiert wurde. Auch wenn es vielleicht wehtut – so geht's:

Java-Anwendungen liegen meist in kompilierten *.class*-Dateien vor, können aber auch in Java-Archiven[4] (*.jar*-Dateien) gespeichert sein. Beide Arten von Dateien können über den Java-Interpreter *java* ausgeführt werden, der unter Windows in der Datei *java.exe* gespeichert ist. Dieser Interpreter erzeugt automatisch ein Konsolenfenster. Java-Anwendungen, die mit Fenstern arbeiten, können Sie auch über den Interpreter *javaw* starten, der kein Konsolenfenster erzeugt. Handelt es sich um ein Java-Archiv, müssen Sie beim Starten das Argument *-jar* mit angeben.

Voraussetzung für die Ausführung einer Java-Anwendung ist, dass die Java-Laufzeitumgebung (JRE) auf dem Rechner installiert ist. Falls dies nicht der Fall ist, können Sie diese unter der Adresse *java.sun.com/j2se/downloads.html* herunterladen. Der Java-Interpreter ist Teil dieser Laufzeitumgebung.

Da normalerweise nicht mit Sicherheit bekannt ist, in welchem Ordner der Java-Interpreter gespeichert ist, wird der Pfad zu diesem häufig in die Umgebungsvariable `Path` aufgenommen. Sie können das Programm *java.exe* bzw. *javaw.exe* also normalerweise ohne Angabe des Pfads aufrufen. Zum Aufrufen verwenden Sie dann wie im Rezept 99 die `Start`-Methode der `Process`-Klasse.

Java-Interpreter arbeiten mit einem so genannten Klassenpfad (Classpath). In diesem Pfad werden Klassen gesucht, die vom Programm verwendet, aber nicht im Programmordner oder in einem der Standard-Java-Ordner gefunden werden. Häufig ist der Klassenpfad in der Umgebungsvariable `Classpath` eingestellt. Beim Aufruf des Java-Interpreters kann der Klassenpfad aber auch über das Befehlszeilenargument *-classpath* mit angegeben werden.

Die folgende Methode `StartJavaApplication` setzt dieses Wissen um und startet eine Java-Anwendung, deren Pfad übergeben wird. Am zweiten Argument kann ein Klassenpfad übergeben werden.

`StartJavaApplication` überprüft zunächst, ob das Java-Programm überhaupt existiert, um undurchsichtige und beim Start über die Windows-Oberfläche bzw. von Fensteranwendungen an der Konsole nur kurz erscheinende Fehlermeldungen des Java-Interpreters zu vermeiden. Dann zerlegt die Methode den Dateinamen der Java-Datei in den Pfad und den eigentlichen Namen. Danach wird überprüft, ob es sich um eine *.jar*- oder eine *.class*-Datei handelt. Bei einer *.jar*-Datei wird das Befehlszeilenargument *-jar* in der Variablen `arguments` abgelegt, die später zum Aufruf des Java-Interpreters verwendet wird. Bei einer *.class*-Datei wird der Dateiname um die Endung gekürzt, da diese bei solchen Dateien nicht mit übergeben werden darf. Für den Fall einer unbekannten Dateiendung erzeugt `StartJavaApplication` eine Ausnahme (wieder um die undurchsichtigen und an der Konsole ausgegebenen Fehlermeldungen des Java-Interpreters zu vermeiden). Die Startargumente werden dann noch um den u. U. übergebenen Klassenpfad und den Dateinamen des Java-Programms erweitert.

Java-Interpreter sollten (bzw. müssen) aus dem Ordner heraus gestartet werden, in dem das Java-Programm gespeichert ist. In allen anderen Fällen resultiert der Java-Fehler, dass eine oder mehrere Klassen nicht gefunden wurden (in Java ausgedrückt ist das eine Ausnahme

4. Normale Java-Anwendungen bestehen üblicherweise aus mehreren Klassendateien. Zum Start einer solchen Anwendung müssen alle Klassendateien vorhanden sein. Ein Java-Archiv ist eine Datei, die mehrere Klassendateien (und eventuell auch andere Dateien) zu einer Datei zusammenfasst und damit die Weitergabe und das Handling von Java-Anwendungen erleichtert.

vom Typ `ClassDefNotFoundError`), wenn der Pfad zum Ordner der Anwendung nicht im Klassenpfad vorkommt. `StartJavaApplication` wechselt deswegen über `Environment.CurrentDirectory` in den Ordner des Java-Programms.

Danach wird der Java-Interpreter unter der Übergabe der ermittelten Befehlszeilenargumente gestartet. Dabei wird am Argument `isWindowApplication` erkannt, ob *java* oder (für Anwendungen mit Fenstern) *javaw* gestartet werden soll. Tritt beim Start eine Ausnahme vom Typ `Win32Exception` auf, wird diese mit einer genaueren Fehlermeldung (als `IOException`) weitergereicht. Damit ist es später wesentlich einfacher, zu erkennen, wo die Ursache für ein Fehlschlagen der Methode zu suchen ist.

`StartJavaApplication` erfordert das Einbinden der Namensräume `System`, `System.Diagnostics`, `System.IO` und `System.ComponentModel`.

```
public static void StartJavaApplication(string filePath, string classPath,
   bool isWindowApplication)
{
   // Überprüfen, ob die Datei existiert
   if (File.Exists(filePath) == false)
   {
      throw new IOException("Das Java-Programm '" + filePath +
         "' existiert nicht");
   }

   // Variable für die Start-Argumente
   string arguments = null;

   // Pfad und Dateiname trennen
   FileInfo fi = new FileInfo(filePath);
   string path = fi.DirectoryName;
   string filename = fi.Name;

   // Argumente zusammensetzen und den Dateinamen bei einer .class-Datei
   // ohne Endung auslesen
   if (fi.Extension == ".jar")
   {
      arguments += " -jar";
   }
   else if (fi.Extension == ".class")
   {
      filename = filename.Substring(0, filename.Length - 6);
   }
   else
   {
      throw new IOException("Unbekannte Dateiendung '" +
         fi.Extension + "'");
   }
```

Listing 198: Methode zum Starten einer Java-Anwendung

```
   // Den Argumenten den Klassenpfad hinzufügen
   if (classPath != null)
   {
      arguments += " -classpath " + classPath;
   }

   // Den Dateinamen zu den Argumenten hinzufügen
   arguments += " " + filename;

   // In den Ordner wechseln, in dem die Datei gespeichert ist
   Environment.CurrentDirectory = path;

   // Den Java-Interpreter starten
   string javaInterpreter = null;
   if (isWindowApplication)
   {
      javaInterpreter = "javaw.exe";
   }
   else
   {
      javaInterpreter = "java.exe";
   }

   try
   {
      Process.Start(javaInterpreter, arguments);
   }
   catch (Win32Exception ex)
   {
      throw new IOException("Fehler beim Starten des " +
         "Java-Interpreters " + javaInterpreter + ": " +
         ex.Message);
   }
}
```

Listing 198: Methode zum Starten einer Java-Anwendung (Forts.)

Das folgende Listing zeigt eine Beispielanwendung der Methode `StartJavaApplication`. Gestartet werden die Programme *hello.class* und *hello.jar*, die im Anwendungsordner in Unterordnern erwartet werden:

```
// Java-Programm in .class-Dateien ohne explizite Angabe des ClassPath starten
Console.WriteLine("Starte hello.class ...");
string javaFileName = Path.Combine(Application.StartupPath,
   @"java\class\hello.class");
```

Listing 199: Starten einer Java-Anwendung in einer class- und einer jar-Version

```
try
{
   ApplicationUtils.StartJavaApplication(javaFileName, null, true);
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}

// Java-Programm in .jar-Datei ohne explizite Angabe des ClassPath starten
Console.WriteLine("Starte hello.jar ...");
javaFileName = Path.Combine(Application.StartupPath,
   @"java\jar\hello.jar");
try
{
   ApplicationUtils.StartJavaApplication(javaFileName, null, true);
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 199: Starten einer Java-Anwendung in einer class- und einer jar-Version (Forts.)

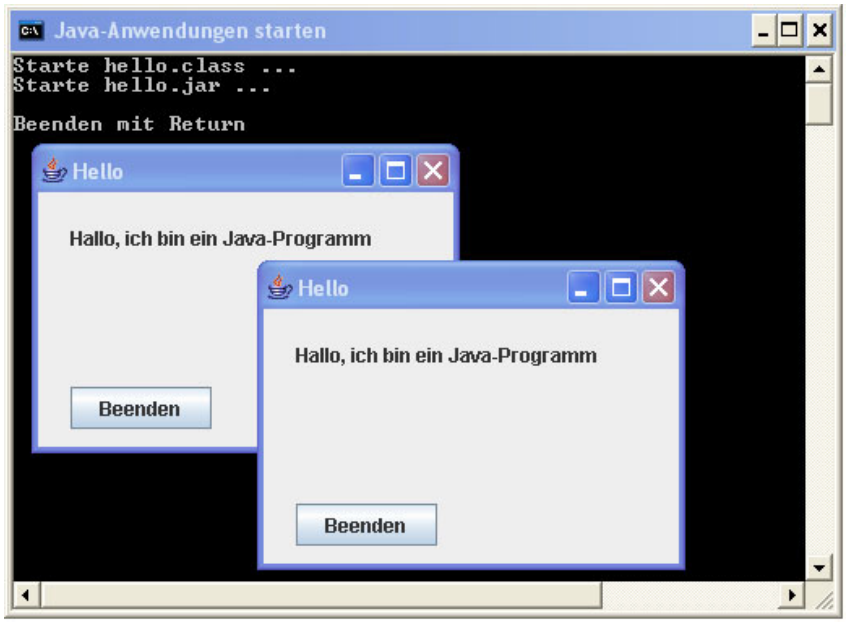


Abbildung 64: Das Beispielprogramm hat zwei (gleichartige) Java-Anwendungen gestartet

Tipp

Für den Fall, dass der Java-Interpreter nicht gefunden wird, können Sie dem Anwender über das Rezept 134 die Möglichkeit geben, diesen automatisch zu suchen. Idealerweise speichern Sie den gefundenen Pfad dann in der Konfigurationsdatei der Anwendung (Rezept 86), um diesen beim nächsten Start dort wieder auszulesen.

105 Die Betätigung der Tastatur simulieren

Sicherlich kennen Sie Anwendungen – meist sind dies Installations-Programme –, die andere Anwendungen starten und wie von Geisterhand bedienen. Fenster öffnen sich, Eingaben werden getätigt, Befehle werden aufgerufen und Fenster werden wieder geschlossen. Und das alles, ohne dass Sie (als Anwender) irgendetwas dazutun.

Solche Anwendungen können Sie auch entwickeln, und das sogar relativ einfach. Besonders dann, wenn Sie andere Anwendungen starten, wie ich es in den Rezepten 99, 100, 101 und 104 beschrieben habe, kann es notwendig sein, dass Sie diese Anwendungen steuern. Die einzige Möglichkeit, die Sie dann besitzen, ist, eine Betätigung der Tastatur zu simulieren. Und das können Sie erreichen, indem Sie die statische `Send`- oder die `SendWait`-Methode der `SendKeys`-Klasse aus dem Namensraum `System.Windows.Forms` aufrufen. Diese Methoden erwarten einen String, der die zu simulierenden Tasten enthält. In diesem String können Sie ganz normale Zeichen angeben, die die Anwendung verarbeiten soll. Für bestimmte (Aktions-)Tasten verwenden diese Methoden jedoch spezielle Zeichenfolgen.

Zeichenfolge	Bedeutung
+	Shift-Taste
^	Strg-Taste
%	Alt-Taste
{BACKSPACE}, {BS} oder {BKSP}	Backspace-Taste
{BREAK}	Pause-Taste
{CAPSLOCK}	Feststelltaste
{DELETE} oder {DEL}	Lösch-Taste (Entf)
{DOWN}	Cursor-nach-unten-Taste
{END}	Ende-Taste
{ENTER} oder ~	Return-(Enter-)Taste
{ESC}	Escape-Taste (Esc)
{HELP}	Hilfe-Taste (F1)
{HOME}	Home-Taste (Pos1)
{INSERT} oder {INS}	Einfügen-Taste (Einfg)
{LEFT}	Cursor-nach-links-Taste
{NUMLOCK}	Num-Lock-Taste
{PGDN}	Bild-nach-unten-Taste

Tabelle 13: Die speziellen Zeichenfolgen der Sendkeys.Send- und der SendWait-Methoden

Zeichenfolge	Bedeutung
{PGUP}	Bild-nach-oben-Taste
{PRTSC}	Druck-Taste (für zukünftige Versionen reserviert)
{RIGHT}	Cursor-nach-rechts-Taste
{SCROLLLOCK}	Rollen-Taste
{TAB}	Tabulator-Taste
{UP}	Cursor-nach-oben-Taste
{F1} ... {F16}	Die Funktionstasten F1 bis F16
{ADD}	Additionstaste auf der Zehnertastatur
{SUBTRACT}	Subtraktionstaste auf der Zehnertastatur
{MULTIPLY}	Multiplikationstaste auf der Zehnertastatur
{DIVIDE}	Divisionstaste auf der Zehnertastatur

Tabelle 13: Die speziellen Zeichenfolgen der Sendkeys.Send- und der SendWait-Methoden (Forts.)

Sie können die normalen und die speziellen Zeichenfolgen beliebig kombinieren. Die Tastenkombination Strg+s definieren Sie z. B. über den String "^s", Alt-F4 geben Sie über "%{F4}" an.

Der Unterschied zwischen `Send` und `SendWait` ist, dass `SendWait` wartet, bis die Tasteneingaben von der Anwendung verarbeitet wurden. Damit kann die startende Anwendung darauf warten, dass Prozesse, die durch die Simulation der Tastaturbetätigung gestartet wurden, beendet wurden. Leider funktioniert dies in vielen Fällen nicht wie gewünscht, was besonders dann gilt, wenn die Anwendung aufgrund der Tasteneingabe einen Bestätigungsdialog anzeigt. Der Prozess, der durch die Tasteneingabe gestartet wurde, ist in diesem Fall nämlich beendet, und `SendKeys` kehrt zurück, während der Dialog noch angezeigt wird. Aber auch wenn `SendWait` in einigen Fällen nicht zum erwarteten Ergebnis führt, ist es dennoch in den meisten Fällen sinnvoller, diese Methode an Stelle von `Send` zu verwenden. Das gilt besonders dann, wenn mehrere Tasteneingaben hintereinander gesendet werden sollen. Unter Umständen ist es manchmal auch sinnvoll, zwischen den Tasteneingaben kleine Pausen einzubauen, damit die andere Anwendung Zeit hat, die Eingaben zu verarbeiten.

Listing 200 zeigt ein Beispiel. Dieses Programm startet den Windows-Editor *Notepad*, schreibt einen Text in die neue Datei, speichert diese und beendet Notepad wieder. Zum Kompilieren dieses Beispiels muss das Projekt die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Windows.Forms`, `System.Diagnostics`, `System.Threading` und `System.IO` importieren.

```
// Notepad starten
Process.Start("Notepad");

// Etwas warten, bis Notepad gestartet ist
Thread.Sleep(500);
```

Listing 200: Simulation von Tasteneingaben am Beispiel des Steuerns von Notepad

```
// Einen kleinen Text schreiben
SendKeys.SendWait("Das ist ein Text, der von .NET kommt");

// STRG+S simulieren
SendKeys.SendWait("^s");

// Überprüfen, ob die zu erzeugende Datei existiert, und
// diese gegebenenfalls löschen
string filename = @"C:\SendKeys_Demo.txt";
if (File.Exists(filename))
{
   File.Delete(filename);
}

// Dateinamen eingeben
SendKeys.SendWait(filename);

// Return simulieren
SendKeys.SendWait("{ENTER}");

// Notepad über ALT+F4 beenden
SendKeys.SendWait("%{F4}");
```

Listing 200: Simulation von Tasteneingaben am Beispiel des Steuerns von Notepad (Forts.)

Das Notepad-Beispiel überprüft bewusst, ob die zu erzeugende Datei bereits existiert, und löscht diese gegebenenfalls. Würde die Datei beim Speichern existieren, würde Notepad einen Bestätigungsdialog anzeigen, um den Anwender zu fragen, ob er die Datei überschreiben möchte. In diesem Fall würde das Programm nicht mehr korrekt funktionieren, denn `SendKeys.SendWait("{ENTER}")` ist bereits komplett abgeschlossen, wenn der Dialog angezeigt wird. Das Programm sendet also direkt danach die Tastenkombination ALT+F4, aber dies, während der Bestätigungsdialog noch angezeigt wird, und somit wirkungslos. Notepad würde nicht beendet werden. Sie sollten also ein wenig mit den Möglichkeiten experimentieren um Anwendungen über die Simulation von Tasteneingaben (relativ) sicher fernsteuern zu können.

Dateisystem

106 Dateiname einer Windowsanwendung ermitteln

Den vollen Dateinamen einer Windowsanwendung (inklusive Pfad) erhalten Sie über die Eigenschaft `ExecutablePath` der `Application`:

```
string filename = System.Windows.Forms.Application.ExecutablePath;
```

Um die `Application`-Klasse verfügbar zu haben, benötigt die Anwendung einen Verweis auf die Assembly *System.Windows.Forms.dll*, was ohne Probleme auch in einer Konsolenanwendung oder Klassenbibliothek möglich ist.

Ohne diesen Verweis können Sie aber auch stattdessen die Eigenschaft `Location` eines `Assembly`-Objekts abfragen, das die »Eintritts-Assembly« (das ist die Assembly, über die das Programm gestartet wurde) repräsentiert. Dieses Objekt erhalten Sie über die Methode `GetEntryAssembly` der `Assembly`-Klasse, die Bestandteil des `System.Reflection`-Namensraums ist:

```
string filename = System.Reflection.Assembly.GetEntryAssembly().Location;
```

Hinweis

Als *Reflektion* (Reflection) werden Features bezeichnet, über die Sie in der Laufzeit eines Programms Informationen über dessen Assemblys, die darin enthaltenen Klassen und deren Schnittstellen, Methoden, Eigenschaften, Felder und Ereignisse etc. auslesen können. Ein weiterer Teil von Reflektion ist das dynamische Erzeugen von Klassen und Assemblys, also das Erzeugen von Programmcode in der Laufzeit eines Programms.

Häufig wird auch vorgeschlagen, die »ausführende Assembly« zu verwenden, die Sie über die Methode `GetExecutingAssembly` erreichen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Dabei müssen Sie allerdings aufpassen: Wird diese Methode in einer Klassenbibliothek aufgerufen, liefert sie ein `Assembly`-Objekt zurück, das die Assembly der *Klassenbibliothek* repräsentiert. Der ausgelesene Dateiname ist also in diesem Fall nicht der der Anwendung, sondern der der Klassenbibliothek.

In allen Fällen erhalten Sie den Dateinamen inklusive Pfad. Wenn Sie nur den Namen der Datei auslesen wollen, können Sie diesen über ein `FileInfo`-Objekt ermitteln. Ein solches Objekt dient dem Auslesen von Informationen zu einer Datei und liefert u. a. über Eigenschaft `Name` den Dateinamen. Die Methode `GetApplicationFilename` in Listing 201 liest den Dateinamen der Anwendung aus und liefert diesen zurück. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

```
public static string GetApplicationFilename()
{
   // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-Assembly
   // speichert
   FileInfo fi = new FileInfo(Assembly.GetEntryAssembly().Location);

   // Dateiname auslesen und zurückgeben
   return fi.Name;
}
```

Listing 201: Methode zum Auslesen des Dateinamens einer Windows- oder Konsolenanwendung

107 Dateiname einer Klassenbibliothek ermitteln

Innerhalb einer Klassenbibliothek können Sie deren Dateinamen ermitteln, indem Sie eine Referenz auf ein `Assembly`-Objekt für die »ausführende Assembly« auslesen und deren `Location`-Eigenschaft abfragen:

```
string filename = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

Wie bereits im vorigen Rezept erhalten Sie den Dateinamen inklusive Pfad. Die Methode `GetLibraryFilename` in Listing 202 liest nur den Dateinamen aus. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

```
public static string GetLibraryFilename()
{
   // FileInfo-Objekt für die Datei erzeugen, die die ausführende
   // Assembly speichert
   FileInfo fi = new FileInfo(Assembly.GetExecutingAssembly().Location);

   // Den Dateinamen auslesen und zurückgeben
   return fi.Name;
}
```

Listing 202: Methode zum Auslesen des Dateinamens einer Klassenbibliothek

Die in die Klassenbibliothek *Addison_Wesley.Codebook.Filesystem.DemoLib.dll* kompilierte Klasse `Demo` in Listing 203 ruft diese Methode in ihrer `GetFilename`-Methode auf:

```
using System;

namespace Addison_Wesley.Codebook.Filesystem.DemoLib
{
   public class Demo
   {
```

Listing 203: Ermittlung des Dateinamens einer Klassenbibliothek

```
      /* Methode zum Auslesen des Dateinamens der Klassenbibliothek.
       * Wird zu Demozwecken von der Anwendung aus aufgerufen */
      public string GetFilename()
      {
         return GetLibraryFilename();
      }
   }
}
```

Listing 203: Ermittlung des Dateinamens einer Klassenbibliothek (Forts.)

Die folgende Konsolenanwendung, die die Klassenbibliothek referenziert, erzeugt eine Instanz der Klasse `Demo` und ermittelt über deren Methode `GetFilename` den Dateinamen der Klassenbibliothek:

```
using System;
using Addison_Wesley.Codebook.Filesystem.DemoLib;

namespace Bibliotheks_Dateiname
{
   class Start
   {
      [STAThread]
      static void Main(string[] args)
      {
         // Instanz der Demo-Klasse in der Klassenbibliothek erzeugen
         Demo d = new Demo();

         // Dateiname der Klassenbibliothek ermitteln
         Console.WriteLine("Dateiname der Klassenbibliothek:\r\n{0}",
            d.GetFilename());
      }
   }
}
```

Listing 204: Konsolenanwendung, die die Methode GetFilename der Bibliotheks-Klasse Demo aufruft

108 Ordner einer Windowsanwendung ermitteln

In einer Anwendung mit Referenz auf die Assembly *System.Windows.Forms.dll* können Sie den Pfad der Anwendung über die Eigenschaft `StartupPath` der `Application`-Klasse ermitteln:

```
string applicationPath = System.Windows.Forms.Application.StartupPath;
```

Ohne Referenz auf die *System.Windows.Forms.dll* können Sie stattdessen auch den vollen Dateinamen der »Eintritts-Assembly« ermitteln und über ein `FileInfo`-Objekt den Pfad auslesen, wie es die Methode `GetApplicationFolderName` in Listing 205 zeigt.

```
public static string GetApplicationFolderName()
{
   // FileInfo-Objekt für die Datei erzeugen, die die Eintritts-Assembly
   // speichert
   FileInfo fi = new FileInfo(Assembly.GetEntryAssembly().Location);

   // Den Pfad des Ordners der Datei zurückgeben
   return fi.DirectoryName;
}
```

Listing 205: Methode zum Auslesen des Ordnernamens einer Windows- oder Konsolenanwendung

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

Beide Varianten funktionieren auch ohne Probleme in einer Klassenbibliothek. Sie erhalten dann allerdings den Ordnernamen der Anwendung zurück, nicht den der Klassenbibliothek.

109 Ordner einer Klassenbibliothek ermitteln

Zur Ermittlung des Ordners einer Klassenbibliothek innerhalb deren Programmcodes können Sie wie beim Auslesen des Dateinamens (Rezept 107) ein `Assembly`-Objekt für die ausführende Assembly erzeugen, dessen `Location`-Eigenschaft auslesen und den Pfad über die `DirectoryName`-Eigenschaft eines `FileInfo`-Objekts extrahieren. Die Methode `GetLibraryFolderName` in Listing 206 nimmt Ihnen diese Arbeit ab. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` einbinden.

```
public static string GetLibraryFolderName()
{
   // FileInfo-Objekt für die Datei erzeugen, die die ausführende
   // Assembly speichert
   FileInfo fi = new FileInfo(Assembly.GetExecutingAssembly().Location);

   // Den Pfad des Verzeichnisses der Datei zurückgeben
   return fi.DirectoryName;
}
```

Listing 206: Methode zum Auslesen des Ordnernamens einer Klassenbibliothek

110 Dateiname für eine temporäre Datei erzeugen

Wenn Sie eine temporäre Datei erzeugen wollen bzw. müssen, sollten Sie für diese einen eindeutigen Dateinamen ermitteln, dessen Pfad dem Systempfad für temporäre Dateien entspricht. Dazu können Sie einfach die Methode `GetTempFileName` der `Path`-Klasse verwenden:

```
string tempFileName = System.IO.Path.GetTempFileName();
```

Sie können absolut sicher sein, dass der ermittelte Dateiname eindeutig ist, d. h. dass noch keine Datei gleichen Namens existiert: `GetTempFileName` ermittelt einen Dateinamen in der Form *tmpnn.tmp*, wobei *nn* für eine hexadezimale Zahl steht, die bei einem im System verwalteten letzten Startwert beginnt. Zur Überprüfung, ob im Temp-Ordner bereits eine Datei gleichen Namens existiert, versucht `GetTempFileName` eine 0-Byte-Datei mit diesem Namen anzulegen. Tritt dabei kein (interner) Fehler auf, ist der Dateiname eindeutig.

Daraus folgt allerdings auch, dass die von `GetTempFileName` erzeugten 0-Byte-Dateien nicht automatisch gelöscht werden und Sie beim Erstellen einer »echten« Datei diese überschreiben müssen.

111 Ordner für temporäre Dateien ermitteln

Den Systemordner für temporäre Dateien erhalten Sie über die Methode `GetTempPath` der `Path`-Klasse:

```
string tempFolder = System.IO.Path.GetTempPath();
```

112 Windows-(Spezial-)Ordner auslesen

Den Windows-Systemordner erhalten Sie recht einfach über die Eigenschaft `SystemDirectory` der Klasse `System.Environment`:

```
string systemDirectory = Environment.SystemDirectory;
```

Den Pfad spezieller Ordner wie z. B. den des Programmordners können Sie über die Methode `GetFolderPath` ermitteln, der Sie eine Konstante der Aufzählung `SpecialFolder` übergeben. Den Ordner für Programme können Sie z. B. folgendermaßen auslesen:

```
string programDirectory = Environment.GetFolderPath(
   Environment.SpecialFolder.ProgramFiles);
```

Die Vielzahl der möglichen Pfade wird in der Hilfe zur `SpecialFolder`-Aufzählung ausführlich erläutert.

Leider können Sie über die `Environment`-Klasse nicht ohne weiteres das Windows-Verzeichnis ermitteln. Dazu können Sie die Umgebungsvariable `windir` abfragen:

```
string windowsDirectory = Environment.GetEnvironmentVariable("windir");
```

Da Umgebungsvariablen auch gelöscht werden können und in neueren Windows-Versionen nicht mehr unbedingt vorhanden sein müssen, sollten Sie stattdessen lieber die API-Funktion `GetWindowsDirectory` verwenden.

Listing 207 kapselt den Aufruf dieser Funktion in die Methode `GetWindowsDirectoryName`. Weitere Methoden geben den Namen der (in meinen Augen ☺) wichtigsten Windows-Systemordner zurück. Die Methoden erfordern den Import der Namensräume `System`, `System.Runtime.InteropServices` und `System.Text`.

```
/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll", SetLastError=true)]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
   uint uSize);

/* Gibt den Pfad zum Windows-Ordner zurück */
public static string GetWindowsDirectoryName()
{
   const int MAX_PATH = 260;
   StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
   if (GetWindowsDirectory(buffer, MAX_PATH + 1) > 0)
   {
      return buffer.ToString();
   }
   else
   {
      throw new Exception("Windows-Fehler " + Marshal.GetLastWin32Error() +
         " beim Ermitteln des Windows-Ordners");
   }
}

/* Gibt den Pfad zum Windows-Systemordner zurück */
public static string GetSystemDirectoryName()
{
   return Environment.SystemDirectory;
}

/* Gibt den Pfad zum Programmordner zurück */
public static string GetProgramDirectoryName()
{
   return Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles);
}

/* Gibt den Pfad zum Ordner für Anwendungsdaten für
  "nicht wandernde" Benutzer zurück */
public static string GetNonRoamingUserApplicationDataDirectoryName()
{
   return Environment.GetFolderPath(
      Environment.SpecialFolder.LocalApplicationData);
}

/* Gibt den Pfad zum Ordner für Anwendungsdaten für
  "wandernde" Benutzer zurück */
public static string GetRoamingUserApplicationDataDirectoryName()
{
```

Listing 207: Methoden zur Ermittlung der Pfade zu den wichtigsten Systemordnern

```
   return Environment.GetFolderPath(
      Environment.SpecialFolder.ApplicationData);
}

/* Gibt den Pfad zum Ordner für Anwendungsdaten für alle Benutzer zurück */
public static string GetAllUsersApplicationDataDirectoryName()
{
   return Environment.GetFolderPath(
      Environment.SpecialFolder.CommonApplicationData);
}

/* Gibt den Pfad zum Desktop-Ordner zurück */
public static string GetDesktopDirectoryName()
{
   return Environment.GetFolderPath(
      Environment.SpecialFolder.DesktopDirectory);
}

/* Gibt den Pfad zum Favoritenordner zurück */
public static string GetFavoritesDirectoryName()
{
   return Environment.GetFolderPath(Environment.SpecialFolder.Favorites);
}

/* Gibt den Pfad zum Ordner zurück, der Verweise zu den zuletzt geöffneten
Dokumenten enthält */
public static string GetRecentDirectoryName()
{
   return Environment.GetFolderPath(Environment.SpecialFolder.Recent);
}
```

Listing 207: Methoden zur Ermittlung der Pfade zu den wichtigsten Systemordnern (Forts.)

Abbildung 65 zeigt eine Konsolenanwendung, die über diese Methoden die Namen der wichtigen Windows-Systemordner ermittelt hat.

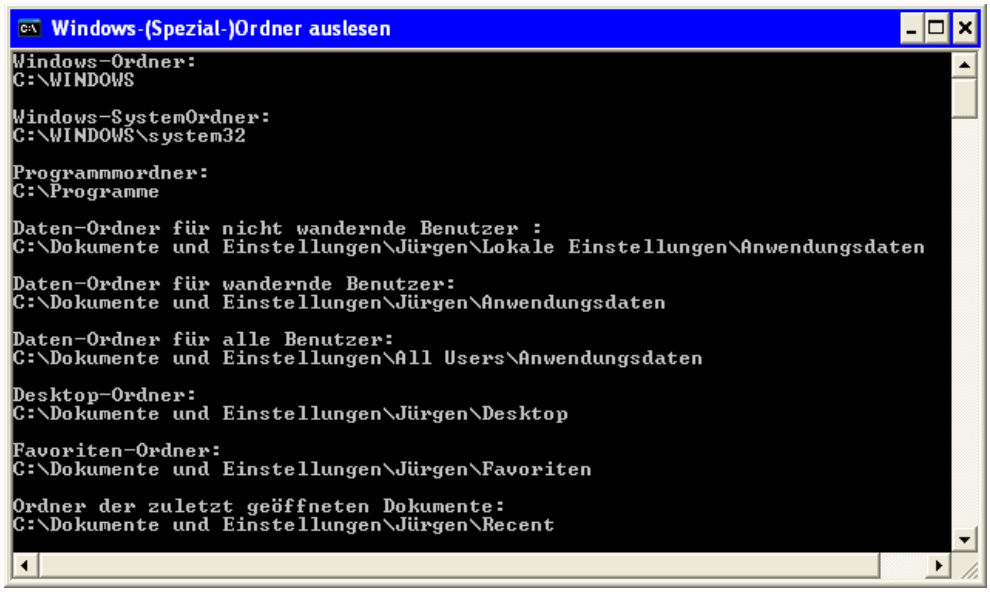


Abbildung 65: Beispielprogramm zur Ermittlung der Windows-Systemordner

113 Das .NET-Framework-Verzeichnis ermitteln

Das Verzeichnis des .NET Framework, das der aktuelle Prozess (also der Prozess, in dem das Programm ausgeführt wird) verwendet, können Sie über die API-Funktion `GetCORSystemDirectory` ermitteln:

```
using System;
using System.Text;
using System.Runtime.InteropServices;

...

/* Deklaration der API-Funktion GetCORSystemDirectory */
[DllImport("mscoree.dll", CharSet = CharSet.Unicode)]
static extern int GetCORSystemDirectory(
  [MarshalAs(UnmanagedType.LPWStr)] StringBuilder pbuffer,
  int cchBuffer, out int dwlength);

/* Gibt den Pfad zum .NET-Framework-Ordner zurück */
public static string GetFrameworkPath()
{
   StringBuilder buffer = new StringBuilder(1024);
   int length;
   if (GetCORSystemDirectory(buffer, buffer.Capacity, out length) == 0)
```

Listing 208: Methode zur Ermittlung des .NET-Framework-Verzeichnisses

```
    {
        return buffer.ToString();
    }
    else
    {
        throw new Exception("Windows-Fehler " + Marshal.GetLastWin32Error() +
            " beim Ermitteln des .NET-Framework-Ordners");
    }
}
```

Listing 208: Methode zur Ermittlung des .NET-Framework-Verzeichnisses (Forts.)

Hinweis

Auch bei .NET-3.5-Anwendungen handelt es sich dabei um das .NET-2.0-Framework-Verzeichnis, da .NET3.5 ja nur auf dem .NET Framework 2.0 aufsetzt und die 2.0 CLR verwendet.

114 Dateiname mit anderer Endung ermitteln

In vielen Fällen, z. B. beim Konvertieren einer Grafikdatei in einen anderen Grafiktyp, werden Sie einen Dateinamen mit einer anderen Dateiendung ermitteln wollen. Die Methode `ChangeExtension` der `Path`-Klasse übernimmt diese Aufgabe.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
string filename = @"c:\Test.jpg";
string changedFilename = Path.ChangeExtension(filename, ".gif");
```

Listing 209: Ermittlung eines Dateinamens mit einer anderen Endung

115 Relativen Pfad aus einem absoluten Pfad ermitteln

Wenn Sie aus einer absoluten Pfadangabe einen relativen Pfad ermitteln müssen, der sich auf einen Referenz-Pfad bezieht, finden Sie dazu leider keine Möglichkeit in den Klassen des .NET Framework. Die für Probleme solcher Art zuständigen Klassen `FileInfo`, `DirectoryInfo` und `Path` bieten keine entsprechenden Methoden oder Eigenschaften.

Sie können aber die API-Funktion `PathRelativePathTo` einsetzen. Dieser Funktion übergeben Sie am ersten Argument einen ausreichend groß dimensionierten `StringBuilder`, der von der Funktion mit dem relativen Pfad beschrieben wird. Am zweiten Argument übergeben Sie den Referenzpfad, am dritten definieren Sie, ob es sich bei diesem Pfad um ein Verzeichnis oder um eine Datei handelt. Handelt es sich um ein Verzeichnis, übergeben Sie die Konstante `FILE_ATTRIBUTE_DIRECTORY` (`0x10`). Für eine Datei übergeben Sie einen beliebigen anderen Wert (also zum Beispiel 0). Das vierte Argument nimmt den absoluten Pfad entgegen, das letzte Argument bestimmt, ob es sich bei diesem um ein Verzeichnis oder eine Datei handelt.

Hinweis

Die Unterscheidung zwischen Verzeichnissen und Dateien ist für die korrekte Berechnung eines relativen Pfades in einigen Fällen sehr wichtig. Ist der absolute Pfad z. B. *C:\Windows\Explorer.exe* und der Referenzpfad *C:\Windows\System32*, resultiert der relative Pfad *..\Explorer.exe*, wenn der Referenzpfad korrekt als Verzeichnis angegeben wird. Wird der Referenzpfad hingegen (unsinnigerweise) als Datei angegeben, resultiert der inkorrekte relative Pfad *.\Explorer.exe*. Den Referenzpfad allerdings als Datei anzugeben, ist eigentlich nicht notwendig, da dieser wohl immer ein Verzeichnis sein wird. Beim absoluten Pfad habe ich keinen Unterschied festgestellt, auch wenn ich Dateiangaben fälschlicherweise als Verzeichnis und Verzeichnisangaben als Datei definiert habe. Microsoft hat diese Möglichkeiten jedoch vorgesehen, und ich halte mich einfach daran.

`PathRelativePathTo` gibt `true` zurück, wenn ein relativer Pfad ermittelt werden konnte. In allen anderen Fällen, die dann auftreten, wenn die Basis (der Laufwerkbuchstabe, der Start-Ordner oder der UNC-Maschinenname) im Referenz- und im absoluten Pfad ungleich ist, gibt diese Funktion `false` zurück.

Die folgende Methode `GetRelativePath` in Listing 210 setzt `PathRelativePathTo` ein um einen relativen Pfad zu ermitteln. Am Argument `absolutePath` übergeben Sie den absoluten Pfad. `absolutePathIsDirectory` bestimmt, ob es sich bei diesem Pfad um ein Verzeichnis handelt. Das Argument `referencePath` bestimmt den Bezugs-Pfad und über `referencePathIsDirectory` geben Sie an, ob dieser Pfad für ein Verzeichnis steht. Für den Fall, dass `PathRelativePathTo` `false` zurückliefert, wirft `GetRelativePath` eine `IOException`.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System.IO`, `System.Text` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration der API-Funktion PathRelativePathTo */
[DllImport("shlwapi.dll", CharSet=CharSet.Auto)]
static extern bool PathRelativePathTo(System.Text.StringBuilder pszPath,
   string pszFrom, uint dwAttrFrom, string pszTo, uint dwAttrTo);

public static string GetRelativePath(string absolutePath,
   bool absolutePathIsDirectory, string referencePath,
   bool referencePathIsDirectory)
{
   // Die Pfadangaben normalisieren für den Fall, dass
   // ..- und .-Angaben enthalten sind
   absolutePath = Path.GetFullPath(absolutePath);
   referencePath = Path.GetFullPath(referencePath);

   const uint FILE_ATTRIBUTE_DIRECTORY = 0x10;
   const int MAX_PATH = 260;
```

Listing 210: Methode zur Ermittlung einer relativen Pfadangabe

```
   // PathRelativePathTo aufrufen
   StringBuilder relativePath = new StringBuilder(MAX_PATH);
   if (PathRelativePathTo(relativePath, referencePath,
      (referencePathIsDirectory ? FILE_ATTRIBUTE_DIRECTORY : 0),
      absolutePath,
      (absolutePathIsDirectory ? FILE_ATTRIBUTE_DIRECTORY : 0)))
   {
      return relativePath.ToString();
   }
   else
   {
      throw new IOException("Der absolute Pfad (" +
         absolutePath + ") und der Referenzpfad (" +
         referencePath + ") besitzen nicht dieselbe Basis");
   }
}
```

Listing 210: Methode zur Ermittlung einer relativen Pfadangabe (Forts.)

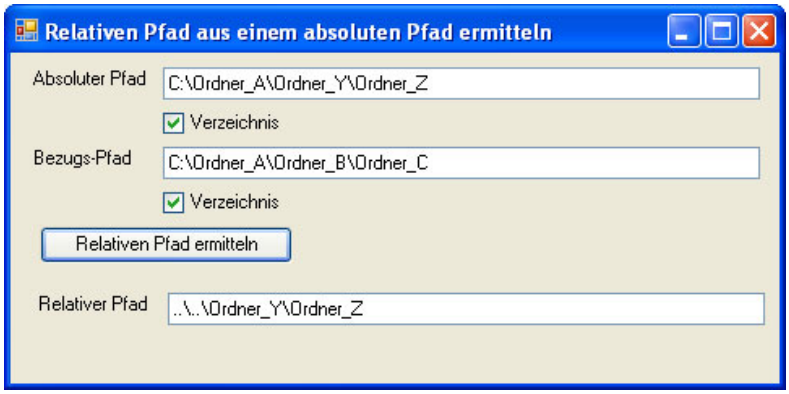


Abbildung 66: Beispielanwendung zur Ermittlung eines relativen Pfads

116 Absoluten Pfad aus einem relativen Pfad ermitteln

Um einen relativen Pfad in einen absoluten Pfad umzuwandeln benötigen Sie lediglich den Bezugs-Pfad. Der relative Pfad *..\Fonts* kann z. B. einfach zu dem Bezugs-Pfad *C:\Windows\System32* addiert werden und ergibt den »absoluten« Pfad *C:\Windows\System32\..\Fonts* (also den eigentlichen Pfad *C:\Windows\Fonts*). Dabei müssen Sie natürlich darauf achten, dass der Bezugs-Pfad mit einem Backslash abgeschlossen ist. Der resultierende Pfad ist für alle Datei-Operation gültig, ist aber u. U. – vor allem für Anwender – nicht besonders übersichtlich. Deswegen sollten Sie diesen Pfad über `Path.GetFullPath` in eine vollständig absolute Pfadangabe umwandeln. Die Methode `GetAbsolutePath` in Listing 211 übernimmt die dafür notwendigen Arbeiten. Zum Kompilieren dieser Methode müssen Sie den Namensraum `System.IO` einbinden.

```
public static string GetAbsolutePath(string relativePath,
   string referencePath)
{
   if (referencePath.EndsWith("\\") == false)
   {
      referencePath += "\\";
   }
   return Path.GetFullPath(referencePath + relativePath);
}
```

Listing 211: Methode zur Umwandlung eines relativen Pfads in einen absoluten

117 Überprüfen, ob eine Pfadangabe gültig ist

Auf Windows-Systemen dürfen Pfadangaben fast alle Zeichen beinhalten. Lediglich einige besondere Zeichen sind nicht erlaubt. So dürfen in Dateinamen zum Beispiel keine Kleiner- und Größer-Zeichen (<, >), Anführungszeichen und keine senkrechten Balken (|) vorkommen. Eine Liste der in einem Pfad ungültigen Zeichen erhalten Sie über die `GetInvalidPathChars`-Methode der `Path`-Klasse. Eine Liste der in Dateinamen ungültigen Zeichen liefert die `GetInvalidFileNameChars`-Methode. Unverständlicherweise ist aber laut der Dokumentation nicht gewährleistet, dass diese Auflistungen alle für Datei- und Verzeichnisnamen ungültigen Zeichen enthalten. Der Grund für diese Unvollständigkeit kann darin liegen, dass unterschiedliche (zukünftige) Dateisysteme eventuell erweiterte Pfadangaben erlauben.

Einen Pfad über das Durchgehen der einzelnen Zeichen zu überprüfen, indem die einzelnen Zeichen mit den Zeichen verglichen werden, die in den von `GetInvalidPathChars` und `GetInvalidFileNameChars` zurückgegebenen Arrays verwaltet werden, wäre also für die Praxis zu unsicher.

Die sicherste Möglichkeit, einen Pfad auf seine Gültigkeit zu überprüfen, wäre, die Datei bzw. den Ordner temporär anzulegen. Tritt dabei kein Fehler auf, ist der Pfad gültig. Diese Lösung ist jedoch nicht praktikabel, da es sein kann, dass der die Anwendung ausführende Benutzer keine Schreibrechte besitzt oder der Pfad temporär (zum Beispiel weil es sich um einen Netzwerkpfad handelt) nicht verfügbar ist. Außerdem müssten Sie dabei beachten, dass es sein kann, dass die Datei bzw. der Ordner bereits existiert, und Sie müssten die temporär angelegte Datei und gegebenenfalls alle angelegten Ordner nach der Überprüfung wieder löschen.

Sie können einen Pfad jedoch recht einfach auf seine Gültigkeit hin überprüfen, indem Sie diesen der `GetFullPath`-Methode der `Path`-Klasse übergeben. Wirft diese Methode eine `NotSupportedException` oder eine `ArgumentException`, ist der Pfad ungültig. Eine `NotSupportedException` wird geworfen, wenn die Pfadangabe zwar prinzipiell gültig ist, aber vom Betriebssystem nicht unterstützt wird (z. B. wenn Doppelpunkte in Ordner- oder Dateinamen enthalten sind). Eine `ArgumentException` wird geworfen, wenn ungültige Zeichen im Pfad enthalten sind.

Die Methode `IsPathValid` in Listing 212 nutzt diese Technik. Sie fängt bewusst nur die beiden genannten Ausnahmen ab um bei eventuell anderen Ausnahmen nicht fälschlicherweise lediglich zurückzumelden, dass der Pfad ungültig ist.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.IO` importieren.


```
public static bool IsPathValid(string path)
{
   try
   {
      // GetFullPath aufrufen um eine Exception
      // bei einem ungültigen Pfad zu provozieren
      Path.GetFullPath(path);
      return true;
   }
   catch (NotSupportedException)
   {
      // Wird bei ungültigen Pfad-Formaten geworfen,
      // z.B. wenn ein Ordner- oder ein Dateiname
      // Doppelpunkte enthält
      return false;
   }
   catch (ArgumentException)
   {
      // Wird bei ungültigen Zeichen im Pfad geworfen
      return false;
   }
}
```

Listing 212: Methode zur Überprüfung der Gültigkeit einer Pfadangabe

Hinweis

`GetAbsolutePath` überprüft lediglich die grundsätzliche Gültigkeit der Pfadangabe. Die Methode überprüft nicht, ob der Pfad auf dem aktuellen System möglich ist. Der Pfad *x:\Demo.txt* ist zum Beispiel gültig, aber nur dann verfügbar, wenn auch ein Laufwerk *x* existiert.

118 Die logischen Laufwerke des Systems ermitteln

Sie können alle logischen Laufwerke eines Systems ermitteln, indem Sie die Methode `GetDrives` der `DriveInfo`-Klasse (aus dem Namensraum `System.IO`) aufrufen. Diese Methode gibt ein Array von `DriveInfo`-Objekten zurück, über die Sie einige Informationen zu den Laufwerken erhalten. Das folgende Beispiel gibt diese Informationen (außer den Laufwerktyp, siehe Rezept 119) aus. Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System` und `System.IO` importieren.

Achtung

Bei der Arbeit mit `DriveInfo` müssen Sie beachten, dass die Eigenschaften `DriveFormat`, `TotalSize`, `TotalFreeSpace`, `AvailableFreeSpace` und `VolumeLabel` nur dann verfügbar sind, wenn das Laufwerk bereit ist (wenn `IsReady true` ergibt).

```
foreach (var driveInfo in DriveInfo.GetDrives())
{
   Console.WriteLine("Name: {0}", driveInfo.Name);
   if (driveInfo.IsReady)
   {
      Console.WriteLine("Label: {0}", driveInfo.VolumeLabel);
      Console.WriteLine("Dateisystem: {0}", driveInfo.DriveFormat);
      Console.WriteLine("Gesamtgröße: {0} Byte", driveInfo.TotalSize);
      Console.WriteLine("Freier Speicher: {0} Byte",
         driveInfo.TotalFreeSpace);
      Console.WriteLine("Für den akt. Benutzer verfügbarer freier " +
         "Speicher: {0} Byte", driveInfo.AvailableFreeSpace);
   }
   else
   {
      Console.WriteLine("Nicht verfügbar");
   }
   Console.WriteLine("Wurzelverzeichnis: {0}",
      driveInfo.RootDirectory.FullName);
   Console.WriteLine();
}
```

Listing 213: Ermitteln von Informationen zu den logischen Laufwerken des Systems

`GetDrives` liefert Informationen zu allen Laufwerken des Systems, neben den Festplattenlaufwerken sind das auch entfernbare (wie Disketten-Laufwerke) und CD-ROM-Laufwerke und solche, die mit Netzwerkordnern verknüpft sind oder die mit dem DOS-Befehl `subst` erzeugt wurden. Abbildung 67 zeigt das Beispielprogramm auf meinem System. A: ist ein Diskettenlaufwerk (ich weiß auch nicht, warum ich das noch habe ...), C: und D: sind Festplattenpartitionen.

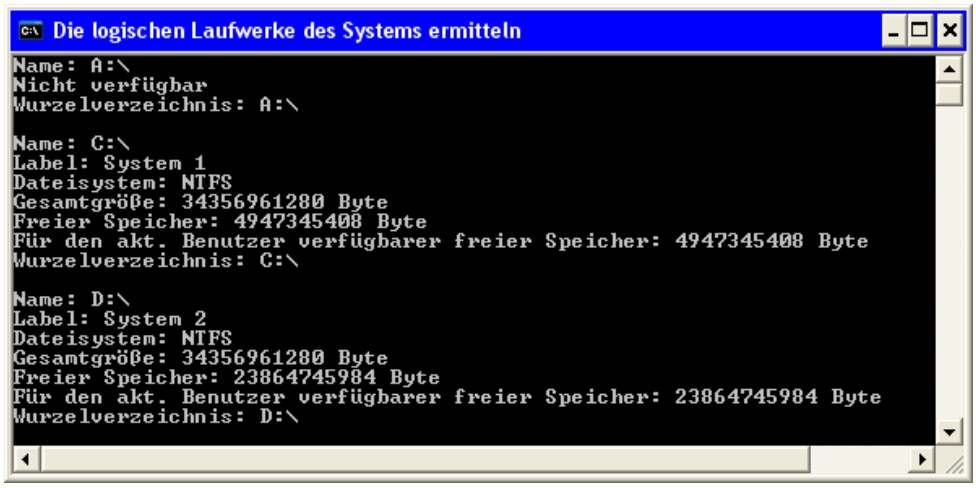


Abbildung 67: Ermittlung aller logischen Laufwerke meines Systems

119 Den Typ der Laufwerke eines Systems ermitteln

Zur Ermittlung der Typen der Laufwerke eines Systems können Sie die Klasse `DriveInfo` aus dem Namensraum `System.IO` verwenden. Bei der Erzeugung einer neuen Instanz übergeben Sie den Namen des Laufwerks. Sie können aber auch, wie ich in Rezept 118 beschrieben habe, die `GetDrives`-Methode der `DriveInfo`-Klasse aufrufen um alle logischen Laufwerke des Systems zu ermitteln.

Die Eigenschaft `DriveType` einer `DriveInfo`-Instanz gibt den Typ des Laufwerks an. Listing 214 zeigt eine beispielhafte Anwendung dieser Klasse. Das Programm gibt Informationen zu allen logischen Systemlaufwerken aus. Um dieses Beispiel kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` importieren.

```
// Alle Laufwerke des Systems durchgehen und den Laufwerk-Typ ermitteln
foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
{
   // Laufwerk-Namen ausgeben
   Console.Write(driveInfo.Name);

   // Den Typ des Laufwerks auslesen
   switch (driveInfo.DriveType)
   {
      case DriveType.CDRom:
         Console.WriteLine("CD/DVD-ROM");
         break;

      case DriveType.Fixed:
         Console.WriteLine("Festplatten-Partition");
         break;

      case DriveType.Ram:
         Console.WriteLine("Ram-Disk");
         break;

      case DriveType.Network:
         Console.WriteLine("Netzwerklaufwerk");
         break;

      case DriveType.NoRootDirectory:
         Console.WriteLine("Laufwerk ohne Stamm-Verzeichnis");
         break;

      case DriveType.Removable:
         Console.WriteLine("Entfernbares Laufwerk");
         break;
```

Listing 214: Ermitteln des Typs aller Laufwerke des Systems

```
      case DriveType.Unknown:
         Console.WriteLine("Unbekannter Typ");
         break;
   }
}
```

Listing 214: Ermitteln des Typs aller Laufwerke des Systems (Forts.)

Hinweis

Falls Sie sich fragen, was ein »Laufwerk ohne Stamm-Verzeichnis« ist: Ich weiß es auch nicht und habe auch im Internet nichts dazu gefunden. Wahrscheinlich handelt es sich dabei um ein Bandlaufwerk.



Abbildung 68: Auslesen von Laufwerk-Informationen in einer Konsolenanwendung

Hinweis

Beachten Sie, dass die in diesem Beispiel nicht verwendeten Eigenschaften `DriveFormat`, `TotalSize`, `TotalFreeSpace`, `AvailableFreeSpace` und `VolumeLabel` nur dann verfügbar sind, wenn das Laufwerk bereit ist (wenn `IsReady` `true` ergibt).

120 Alle Laufwerke eines bestimmten Typs ermitteln

Zur Ermittlung aller Laufwerke eines Systems, die einen bestimmten Typ besitzen, habe ich eine Methode geschrieben, die wie Rezept 119 die `DriveInfo`-Klasse aus dem Namensraum `System.IO` einsetzt. Diese Methode gibt eine Instanz der generischen `List`-Klasse typisiert mit dem Datentyp `DriveInfo` zurück.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Collections.Generic` einbinden.

```
public static List<DriveInfo> GetDriveInfos(DriveType driveType)
{
   // Ergebnis-Liste instanzieren
   List<DriveInfo> driveInfos = new List<DriveInfo>();

   // Alle Laufwerke des Systems durchgehen und den Laufwerk-Typ ermitteln
   foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
   {
      if (driveInfo.DriveType == driveType)
      {
         driveInfos.Add(driveInfo);
      }
   }

   // Die Liste zurückgeben
   return driveInfos;
}
```

Listing 215: Methode zur Ermittlung aller Laufwerke eines bestimmten Typs

Das folgende Beispiel zeigt eine Anwendung dieser Methode in einer einfachen Konsolenanwendung.

```
// Festplatten-Partitionen ermitteln ...
Console.WriteLine("Alle Festplatten-Partitionen:");
List<DriveInfo> driveInfos = GetDriveInfos(DriveType.Fixed);

// ...und durchgehen
foreach (DriveInfo driveInfo in driveInfos)
{
   Console.WriteLine(driveInfo.Name.Replace(":\\", ""));
}

// CD-ROM-Laufwerke ermitteln ...
Console.WriteLine("\r\nAlle CD-ROM-Laufwerke:");
driveInfos = GetDriveInfos(DriveType.CDRom);

// ...und durchgehen
foreach (DriveInfo driveInfo in driveInfos)
{
   Console.WriteLine(driveInfo.Name.Replace(":\\", ""));
}
```

Listing 216: Beispielanwendung der GetDriveInfo-Methode

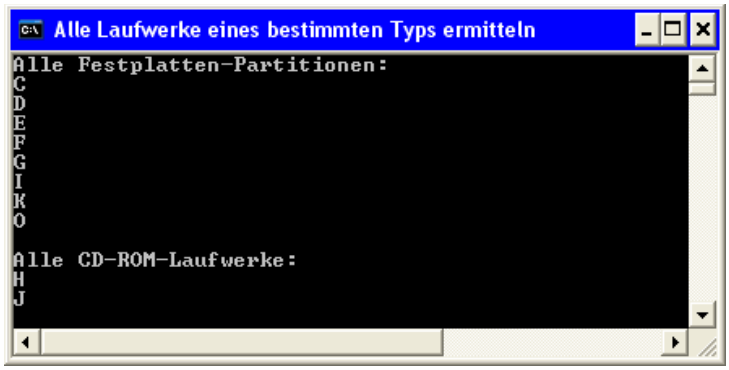


Abbildung 69: Die Beispielanwendung zur Ermittlung der Festplatten-Partitionen und der CD-ROM-Laufwerke in Aktion

121 Größe und freien Speicherplatz eines Laufwerks ermitteln

Zur Ermittlung der Größe und des freien Speicherplatzes eines Laufwerks können Sie, wie schon bei der Ermittlung der Typen der Laufwerke eines Systems (Rezept 119), eine Instanz der Klasse `DriveInfo` verwenden. Die Eigenschaft `TotalSize` liefert die Gesamtgröße des Laufwerks in Byte. Über die Eigenschaft `TotalFreeSpace` erhalten Sie eine Information über den insgesamt freien Speicherplatz. Die Eigenschaft `AvailableFreeSpace` liefert schließlich den freien Platz für den Benutzer, unter dessen Konto die Anwendung ausgeführt wird. Diese Angabe kann kleiner ausfallen als die des insgesamt freien Speicherplatzes, da es möglich sein kann, dass den einzelnen Benutzern auf dem angegebenen Laufwerk Speicherplatz-Quoten zugeteilt wurden.

Das folgende Beispiel zeigt eine Anwendung dieser Eigenschaften. Zum Kompilieren müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
Console.WriteLine("Speicherplatz von C:");

// DriveInfo-Instanz für das Laufwerk C: erzeugen
DriveInfo driveInfo = new DriveInfo("C:");

// Größe und freien Speicherplatz ausgeben
Console.WriteLine("Größe: {0} Byte", driveInfo.TotalSize);
Console.WriteLine("Gesamter freier Platz: {0} Byte",
    driveInfo.TotalFreeSpace);
Console.WriteLine("Für den aktuellen Benutzer verfügbarer " +
   "freier Platz: {0} Byte", driveInfo.AvailableFreeSpace);
```

Listing 217: Ermittlung der Größe und des freien Speicherplatzes eines Laufwerks



Abbildung 70: Das Beispielprogramm zum Auslesen von Laufwerk-Größeninformationen

122 Überprüfen, ob ein Ordner existiert

Mit .NET-Features können Sie zwei Wege gehen um zu überprüfen, ob ein Ordner existiert. Der einfachste Weg ist, dazu die statische `Exists`-Methode der `Directory`-Klasse aus dem Namensraum `System.IO` zu verwenden:

```
string folderName = "c:\\temp";
if (Directory.Exists(folderName))
{
   // Der Ordner existiert
   Console.WriteLine(folderName + " existiert");
}
else
{
   // Der Ordner existiert nicht
   Console.WriteLine(folderName + " existiert nicht");
}
```

Alternativ können Sie auch ein DirectoryInfo-Objekt (ebenfalls aus dem Namensraum System.IO) erzeugen und dessen Exists-Eigenschaft abfragen:

```
string folderName = "c:\\temp";
DirectoryInfo di = new DirectoryInfo(folderName);
if (di.Exists)
{
   // Der Ordner existiert
   Console.WriteLine(folderName + " existiert");
}
else
{
   // Der Ordner existiert nicht
   Console.WriteLine(folderName + " existiert nicht");
}
```

Beide Alternativen funktionieren auch ohne Probleme mit Netzwerkordnern, sofern der in Windows eingeloggte Benutzer entsprechende Zugriffsrechte für die Netzwerkressource besitzt.

123 Ordner erzeugen

Ordner können Sie sehr einfach über die Methode `CreateDirectory` der Klasse `System.IO.Directory` erzeugen. Der ersten Variante dieser Methode übergeben Sie dazu einen kompletten Pfadnamen. Sie können damit auch Ordner erzeugen, deren übergeordnete Ordner noch nicht existieren.

`CreateDirectory` existiert übrigens noch in einer zweiten Variante, der Sie am zweiten Argument ein `DirectorySecurity`-Objekt übergeben können. Über dieses Objekt bestimmen Sie die Zugriffsrechte auf diesen Ordner.

`CreateDirectory` führt immer zum Erfolg, wenn die Pfadangabe syntaktisch in Ordnung ist, auch wenn der angegebene Ordner bereits existiert. Für den Fall, dass die Pfadangabe nicht korrekt ist, fängt der folgende Programmcode, der den Ordner *C:\Test\Test\Test* erzeugt, die dann erzeugte Ausnahme ab. Das Programm erfordert die Einbindung der Namensräume `System` und `System.IO`.

```
string folderName = @"C:\Test\Test\Test";
try
{
   Directory.CreateDirectory(folderName);
   Console.WriteLine("Der Ordner '" + folderName +
      "' existierte bereits oder wurde erzeugt");
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Erzeugen des Ordners '" +
      folderName + "': " + ex.Message);
}
```

Listing 218: Erzeugen eines Ordners

124 Ordner mit .NET-Features kopieren

Das .NET Framework bietet nach meinen Recherchen auch in der Version 2.0 keine direkte Möglichkeit, einen Ordner mit Inhalt zu kopieren. Wollen Sie dazu .NET-Features verwenden, müssen Sie den Zielordner zunächst selbst erzeugen. Dann gehen Sie rekursiv den Quellordner und dessen Unterordner durch, erzeugen entsprechende Unterordner im Zielordner und kopieren die einzelnen Dateien in den jeweiligen Ordner. Die Methode `CopyFolder` in Listing 219 implementiert ein solches Kopieren.

`CopyFolder` überprüft zunächst, ob der angegebene Zielordner bereits existiert, und erzeugt in diesem Fall eine Ausnahme vom Typ `IOException` um zu verhindern, dass die im Zielordner bereits vorhandenen Dateien beim Kopieren mit den neuen Dateien unkontrolliert vermischt werden. Wenn Sie das Mischen vorhandener mit neuen Dateien ermöglichen wollen, können Sie die Überprüfung auch weglassen.

Danach erzeugt `CopyFolder` über `System.IO.Directory.CreateDirectory` den Zielordner, allerdings ohne dessen Unterordner. Diese werden in der privaten Methode `CopySubFoldersAndFiles`

erzeugt. CopySubFoldersAndFiles, der ein DirectoryInfo-Objekt und der Name des Quell- und des Zielordners übergeben wird, geht zunächst alle Unterordner des jeweiligen Ordners durch. Diese werden über die Methode GetDirectories des aktuellen Ordners ermittelt. GetDirectories gibt ein Array mit DirectoryInfo-Objekten zurück, das dann in einer for-Schleife durchlaufen wird. Innerhalb dieser Schleife ermittelt CopySubFoldersAndFiles zunächst den Pfad des Ziel-Unterordners, indem der Pfad des Quellordners einfach durch den Pfad des Zielordners ersetzt wird. Danach erzeugt die Methode den jeweiligen Zielordner und ruft sich selbst unter Übergabe eines DirectoryInfo-Objekts für den Zielordner auf um dessen Unterordner zu erzeugen und die enthaltenen Dateien zu kopieren.

Nach dem Durchgehen der Unterordner kopiert CopySubFoldersAndFiles die im jeweiligen Ordner enthaltenen Dateien, wobei der Zieldateiname wieder über das Ersetzen des Quellordnernamens durch den Zielordnernamen ermittelt wird.

Tritt beim Erzeugen eines Ordners oder beim Kopieren einer Datei eine Ausnahme auf, wird diese einfach an den Aufrufer weitergegeben, da Ausnahmen innerhalb der Methoden CopySubFoldersAndFiles und CopyFolder nicht abgefangen werden. Deshalb bricht die CopyFolder-Methode sofort ab, sobald das Erzeugen eines Ordners oder das Kopieren einer Datei fehlschlägt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume System und System.IO einbinden.

```
/* Kopiert einen Ordner */
public static void CopyFolder(string sourceFolderPath, string destFolderPath)
{
   // DirectoryInfo-Objekt für den Quellordner erzeugen
   DirectoryInfo sourceFolder = new DirectoryInfo(sourceFolderPath);

   // Überprüfen, ob der Zielordner bereits existiert
   if (Directory.Exists(destFolderPath))
   {
      // Ausnahme erzeugen
      throw new IOException("Der Zielordner '" + destFolderPath +
         "' existiert bereits");
   }

   // Zielordner anlegen
   Directory.CreateDirectory(destFolderPath);

   // Methode zum Kopieren der Unterordner und Dateien aufrufen
   CopySubFoldersAndFiles(sourceFolder, sourceFolderPath, destFolderPath);
}

/* Kopiert (rekursiv) alle Unterordner und Dateien
  eines Ordners in einen anderen Ordner */
private static void CopySubFoldersAndFiles(DirectoryInfo folder,
```

Listing 219: Methoden zum rekursiven Kopieren eines Ordners

```
   string sourceFolderPath, string destFolderPath)
{
   // Alle Unterordner des übergebenen Ordners durchgehen
   DirectoryInfo[] subFolders = folder.GetDirectories();
   for (int i = 0; i < subFolders.Length; i++)
   {
      // Pfad für den Ziel-Unterordner ermitteln, indem der Pfad zum
      // Quellordner durch den Pfad zum Zielordner ersetzt wird
      string destSubFolderName = subFolders[i].FullName.Replace(
         sourceFolderPath, destFolderPath);

      // Unterordner im Zielordner erzeugen
      Directory.CreateDirectory(destSubFolderName);

      // Funktion rekursiv aufrufen um zunächst die weiteren Unterordner
      // zu erzeugen
      CopySubFoldersAndFiles(subFolders[i], sourceFolderPath,
         destFolderPath);
   }

   // Die im Ordner enthaltenen Dateien ermitteln
   FileInfo[] files = folder.GetFiles();

   // Alle Dateien durchgehen
   for (int i = 0; i < files.Length; i++)
   {
      // Ziel-Dateiname ermitteln, indem der Pfad zum Quellordner
      // durch den Pfad zum Zielordner ersetzt wird
      string destFileName = files[i].FullName.Replace(
         sourceFolderPath, destFolderPath);

      // Datei kopieren
      File.Copy(files[i].FullName, destFileName);
   }
}
```

Listing 219: Methoden zum rekursiven Kopieren eines Ordners (Forts.)

Hinweis

Da der Abbruch beim Fehlschlagen eines Kopiervorgangs in einigen Fällen nicht gewünscht ist, implementiert das Rezept 126 eine Methode, die beim Fehlschlagen einer Dateioperation die Ausnahme abfängt, den Fehler protokolliert und mit den weiteren Dateioperationen fortfährt.

125 Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage über SHFileOperation kopieren

Zum Kopieren eines kompletten Ordners können Sie neben den .NET-Features auch die Windows-API-Funktion `SHFileOperation` verwenden. Diese Funktion steht ab Windows 95 zur Verfügung.

`SHFileOperation` erlaubt das Kopieren, Verschieben, Umbenennen und Löschen von Ordnern und Dateien. Beim Kopieren eines Ordners wird automatisch der gesamte Inhalt kopiert. Dabei zeigt `SHFileOperation` beim Kopieren größerer Inhalte automatisch einen Fortschrittsdialog an. Über Flags können Sie festlegen, ob die Dateioperation rückgängig gemacht werden kann und der Anwender vor dem Überschreiben vorhandener Ordner gefragt wird, ob diese überschrieben werden sollen. Aus einem speziellen Feld der zu übergebenden Struktur können Sie nach dem Kopieren auslesen, ob der Anwender den Kopiervorgang unterbrochen hat.

Die standardmäßige Arbeitsweise von `SHFileOperation` entspricht dem Kopieren von Ordnern über den Windows Explorer (bzw. den Arbeitsplatz). Diese Funktion ermöglicht damit das komfortable und einfache Kopieren ganzer Ordner.

`SHFileOperation` erwartet als Argument eine als Referenz zu übergebende, normale (C-)Struktur mit Informationen zu der auszuführenden Aktion. In dieser Struktur, die im Windows-API mit `SHFILEOPSTRUCT` bezeichnet wird, geben Sie die auszuführende Aktion und die Datei- bzw. Ordnernamen an. In einem »Flags«-Feld können Sie die auszuführende Aktion näher spezifizieren. Für das Kopieren eines Ordners können Sie hier z. B. festlegen, dass der Vorgang rückgängig gemacht werden kann und dass Windows vor dem Überschreiben vorhandener Ordner beim Anwender nachfragt.

Die `SHFILEOPSTRUCT`-Struktur sieht in C# folgendermaßen aus:

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
   public IntPtr hwnd;
   public uint wFunc;
   public string pFrom;
   public string pTo;
   public short fFlags;
   [MarshalAs(UnmanagedType.Bool)]
   public bool fAnyOperationsAborted;
   public IntPtr hNameMappings;
   public string lpszProgressTitle;
}
```

Das `StructLayout`-Attribut legt zunächst fest, dass Elemente der Struktur beim Export in den nicht von der CLR verwalteten Speicher der (in C oder C++ geschriebenen) API-Funktion sequenziell im Speicher angelegt werden, wie es bei klassischen C-Strukturen der Fall ist. Das `CharSet`-Feld legt fest, dass Zeichenketten innerhalb der Struktur automatisch, je nach Betriebssystem, als Unicode oder ASCII exportiert werden.

Innerhalb der Struktur sind die einzelnen Felder genauso deklariert, wie diese von der API-Funktion erwartet werden. Tabelle 14 beschreibt die Bedeutung der einzelnen Felder.

Feld	Bedeutung
hwnd	Hier können Sie den Handle eines Fensters eintragen, in dem SHFileOperation den Fortschritt beim Kopieren oder Verschieben ausgeben soll.
wFunc	bestimmt die Art der auszuführenden Dateioperation mit den folgenden int-Konstanten: ▶ FO_MOVE (0x0001): Verschieben ▶ FO_COPY (0x0002): Kopieren ▶ FO_DELETE (0x0003): Löschen ▶ FO_RENAME (0x0004): Umbenennen
pFrom	Dieses Feld wird mit einer oder mehreren Datei- oder Ordnerangaben belegt, die die Quelldateien bzw. -ordner festlegen. Mehrere Pfadangaben müssen durch ein 0-Zeichen voneinander getrennt werden. Am Ende muss ein zusätzliches 0-Zeichen stehen.
pTo	Dieses Feld verwaltet beim Kopieren, Verschieben und Umbenennen ähnlich dem pFrom-Feld Pfadangaben zu Zielordner bzw. Zieldateien. Beim Löschen von Dateien oder Ordnern muss dieses Feld leer bleiben (also einen Leerstring oder null speichern, was per Voreinstellung der Fall ist).
fFlags	In fFlags können Sie mit festgelegten Flag-Konstanten spezielle Einstellungen zur ausgewählten Dateioperation festlegen. Die einzelnen Konstanten müssen dazu Oder-verknüpft werden (mit |). Die verwendeten Konstanten finden Sie in Tabelle 15.
fAnyOperationsAborted	In diesem Feld legt die SHFileOperation-Funktion true ab, wenn der Anwender die Operation abgebrochen hat.
hNameMappings	Dieses Feld können Sie mit dem Handle eines Name-Mapping-Objekts belegen, in das SHFileOperation beim Kopieren/Verschieben automatisch umbenannte Dateien einträgt, wenn die Flags FOF_RENAMEONCOLLISION und FOF_WANTMAPPINGHANDLE gesetzt sind.
lpszProgressTitle	In diesem Feld können Sie den Titel des Fortschritt-Fensters festlegen, wenn Flags den Wert FOF_SIMPLEPROGRESS enthält.

Tabelle 14: Die Felder der Struktur SHFILEOPSTRUCT

Flag	Bedeutung
FOF_ALLOWUNDO (0x0040)	Rückgängigmachen ermöglichen
FOF_CONFIRMMOUSE (0x0002)	Zurzeit nicht in Verwendung
FOF_FILESONLY (0x0080)	*.*-Angaben nur auf Dateien anwenden
FOF_MULTIDESTFILES (0x0001)	gibt an, dass pTo anstatt eines Zielordners mehrere Zieldateien verwaltet.
FOF_NOCONFIRMATION (0x0010)	Benutzer vor dem Überschreiben von Dateien/Ordnern nicht fragen
FOF_NOCONFIRMMKDIR (0x0200)	Keine Bestätigung für zu erzeugende Ordner anfordern
FOF_NO_CONNECTED_ELEMENTS (0x2000)	Verbundene Dateien nicht als Gruppe verschieben. Gilt nur mit installiertem Internet Explorer ab Version 5.

Tabelle 15: Die Flags der SHFILEOPSTRUCT-Struktur

Flag	Bedeutung
FOF_NOCOPYSECURITYATTRIBS (0x0800)	Sicherheits-Attribute der Datei nicht mit kopieren. Gilt nur mit installiertem Internet Explorer ab Version 4.71.
FOF_NOERRORUI (0x0400)	Keinen Fehlerdialog bei Fehlern anzeigen
FOF_NORECURSION (0x1000)	Nicht rekursiv in Unterordner verzweigen
FOF_RENAMEONCOLLISION (0x0008)	Dateien, die bereits existieren, automatisch neu benennen
FOF_SILENT (0x0004)	Fortschrittsdialog nicht anzeigen
FOF_SIMPLEPROGRESS (0x0100)	Im Fortschrittsdialog keine Dateinamen anzeigen
FOF_WANTMAPPINGHANDLE (0x0020)	Wenn FOF_RENAMEONCOLLISION angegeben ist und Dateien umbenannt werden, steht dieses Flag dafür, dass die umbenannten Dateien in einem Mapping-Objekt festgehalten werden, das im Feld hNameMappings referenziert wird.
FOF_WANTNUKEWARNING (0x4000)	Warnung ausgeben, falls eine Datei beim Löschen zerstört wird. Gilt nur mit installiertem Internet Explorer ab Version 5.

Tabelle 15: Die Flags der SHFILEOPSTRUCT-Struktur (Forts.)

Da der originale BOOL-Datentyp des Feldes fAnyOperationsAborted in der API-Funktion einen anderen Speicherbereich verwendet als der C#-bool-Typ, muss dieser beim Übergeben der Struktur in entsprechende Speicherstruktur umgewandelt werden. Dafür steht das MarshalAs-Attribut, das die Umwandlung von C#-Datentypen bei der Übergabe bestimmt. Das Feld fAnyOperationsAborted wird damit in einen 4-Byte-Boolean-Wert konvertiert, bei dem true ungleich 0 und false gleich 0 ist.

Das Kopieren mit Hilfe von SHFileOperation ist nun recht einfach. Zunächst erzeugen Sie eine Instanz der SHFILEOPSTRUCT-Struktur, definieren in dieser Struktur die Funktion, den Quell- und den Zielordner und setzen die Flags so, dass die Aktion rückgängig gemacht werden kann und dass – je nach Wunsch – der Anwender das Überschreiben vorhandener Ordner bestätigen muss. Dann rufen Sie SHFileOperation unter Übergabe dieser Struktur auf. Die Rückgabe von SHFileOperation ist laut der Dokumentation 0, wenn die Operation erfolgreich war. In Fehlerfall zeigt die Funktion (normalerweise) automatisch einen Fehlerdialog an, wenn das Feld fFlags nicht die Konstante FOF_NOERRORUI (0x0400) oder FOF_SILENT (0x0004) enthält, und gibt einen (leider nicht dokumentierten) Wert ungleich 0 zurück.

Achtung

Bei meinen Versuchen unter Windows XP und Windows 2000 meldete SHFileOperation keinen Fehler beim Versuch, einen Ordner auf ein Laufwerk zu kopieren, das gar nicht existierte (z. B. den Ordner *C:\Projekte* nach *X:\Backup\Projekte*). Auch der Rückgabewert war 0, was es so scheinen lässt, als wäre der Ordner kopiert worden. Nur wenn das Laufwerk vorhanden war, aber der Quell- oder Zielordner nicht existierte, generierte SHFileOperation einen Fehler. Außerdem führt der Aufruf von SHFileOperation zu einem Fehler, wenn der Quell- oder der dem Zielordner übergeordnete Ordner nicht existieren. Leider ist es nicht sinnvoll, diese automatischen und leider schwer verständlichen Fehlermeldungen[1] von SHFileOperation abzuschalten (wie ich am Ende dieses Rezepts erläutere).

1. Schwer verständlich ist z. B. der Fehler »Datei kann nicht kopiert werden. Die Quelldatei oder vom Quelldatenträger kann nicht gelesen werden« beim Versuch, einen nicht existierenden Ordner zu kopieren.

Um diese Probleme zu vermeiden, fragt die Methode `CopyFolder` in Listing 220, die das Kopieren über `SHFileOperation` implementiert, vor dem Kopieren ab, ob der Quell- und der Zielordner existieren, und generiert im negativen Fall eine Ausnahme vom Typ `IOException`.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Runtime.InteropServices` einbinden.

```
/* Deklaration der Windows-API-Funktion SHFileOperation */
[DllImport("shell32.dll", CharSet = CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

/* Konstanten für SHFileOperation (aus ShellAPI.h) */
// Datei oder Ordner kopieren
private const int FO_COPY = 0x0002;
// Rückgängigmachen erlauben
private const int FOF_ALLOWUNDO = 0x0040;
// keine Nachfrage beim Anwender
private const int FOF_NOCONFIRMATION = 0x0010;

/* Struktur zur Definition der Dateioperation */
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
   public IntPtr hwnd;
   public int wFunc;
   public string pFrom;
   public string pTo;
   public short fFlags;
   [MarshalAs(UnmanagedType.Bool)]
   public bool fAnyOperationsAborted;
   public IntPtr hNameMappings;
   public string lpszProgressTitle;
}

/* Kopiert einen Ordner per API-Funktion */
public static void CopyFolder(string sourceFolderPath,
   string destFolderPath, bool confirmOverwrites)
{
   // Überprüfen, ob der Zielordner existiert,
   // um zum einen das Problem zu vermeiden, dass SHFileOperation beim
   // Kopieren auf ein nicht existierendes Laufwerk ohne Fehler
   // ausgeführt wird, und zum anderen den Fehler zu vermeiden, den
```

Listing 220: Methode zum rekursiven Kopieren eines Ordners über SHFileOperation mit Fortschrittsdialog und Undo-Option

```
// SHFileOperation meldet, wenn dieser Ordner nicht existiert.
if (Directory.Exists(destFolderPath) == false)
{
   // Ziel-Parent-Ordner existiert nicht: Ausnahme werfen
   throw new IOException("Der Ziel-Ordner " + destFolderPath +
      " existiert nicht");
}

// Überprüfen, ob der Quellordner existiert
if (Directory.Exists(sourceFolderPath) == false)
{
   throw new IOException("Der Quell-Ordner " + sourceFolderPath +
      " existiert nicht");
}

// Struktur für die Dateiinformationen erzeugen
SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

// (Unter-)Funktion definieren (ShFileOperation kann auch Dateien und
// Ordner löschen, verschieben oder umbenennen)
fileOp.wFunc = FO_COPY;

// Quelle und Ziel definieren. Dabei müssen mehrere Datei- oder
// Ordnerangaben über 0-Zeichen getrennt werden. Am Ende muss ein
// zusätzliches 0-Zeichen stehen
fileOp.pFrom = sourceFolderPath + "\x0\x0";
fileOp.pTo = destFolderPath + "\x0\x0";

// Flags setzen, sodass ein Rückgängigmachen möglich ist und
// dass - je nach Argument confirmOverwrites - keine Nachfrage
// beim Überschreiben von Ordnern beim Anwender erfolgt
if (confirmOverwrites)
{
   fileOp.fFlags = FOF_ALLOWUNDO;
}
else
{
   fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;
}

// ShFileOperation unter Übergabe der Struktur aufrufen
int result = SHFileOperation(ref fileOp);

// Erfolg auswerten. SHFileOperation liefert
// 0 zurück, wenn kein Fehler aufgetreten ist, ansonsten einen
```

Listing 220: Methode zum rekursiven Kopieren eines Ordners über SHFileOperation mit Fortschrittsdialog und Undo-Option (Forts.)

```
   // (leider undokumentierten) Wert ungleich 0.
   if (result != 0)
   {
      throw new IOException("Fehler " + result +
         " beim Kopieren des Ordners '" +
         sourceFolderPath + "' in den Ordner '" +
         destFolderPath + "'");
   }
}
```

Listing 220: Methode zum rekursiven Kopieren eines Ordners über SHFileOperation mit Fortschrittsdialog und Undo-Option (Forts.)

Hinweis

Bei der Anwendung der `CopyFolder`-Methode müssen Sie beachten, dass die Angabe des Zielordners für `SHFileOperation` der Ordner ist, der den zu kopierenden Ordner aufnehmen soll. Wenn Sie den Ordner *C:\Codebook* also in den Ordner *G:\Backup* als Unterordner *Codebook* kopieren wollen, müssen Sie in etwa den folgenden Quelltext verwenden:

```
// Ordnernamen festlegen
string sourceFolderName = @"C:\Codebook";
string destFolderName = @"G:\Backup";

Console.WriteLine("Kopiere {0} nach {1} ...", sourceFolderName,
   destFolderName);

// Zielordner erzeugen, falls dieser nicht vorhanden ist
Directory.CreateDirectory(destFolderName);

// Ordner rekursiv mit Anwender-Nachfrage für das Überschreiben von
// Ordnern kopieren
try
{
   CopyFolder(sourceFolderName, destFolderName, true);
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 221: Kopieren eines Ordners mit der CopyFolder-Methode

Abbildung 71 zeigt diese Anwendung. Da der Zielordner bereits existiert, fragt `SHFileOperation` den Anwender, ob er die eventuell vorhandenen Dateien überschreiben will. Abbildung 72 wird der Ordner gerade kopiert, Abbildung 73, zeigt die Fehlermeldung beim Versuch eine Datei zu kopieren, die gerade in Verwendung ist.

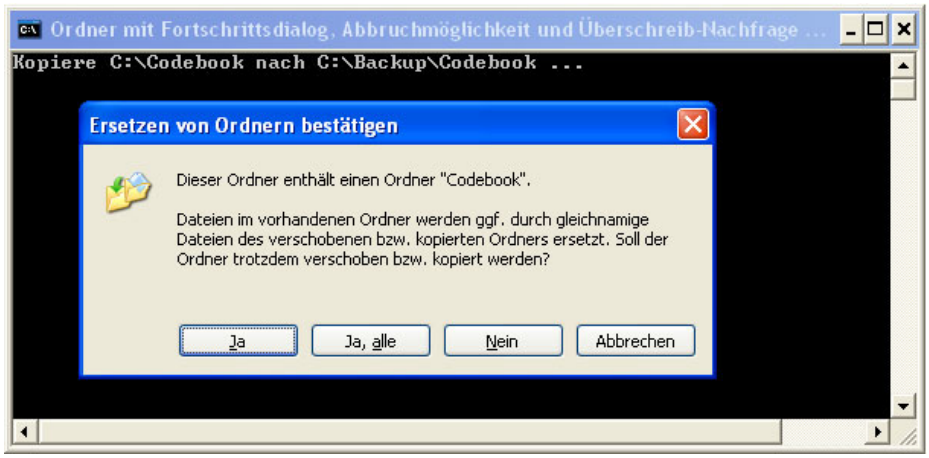


Abbildung 71: Das Beispielprogramm beim Kopieren eines Ordners, der auf dem Zielsystem bereits existiert



Abbildung 72: Das Beispielprogramm beim Kopieren eines Ordners

Hinweis

Im Fehlerfall bricht `SHFileOperation` die Dateioperation ab. Wenn beim Kopieren eines Ordners also eine Datei nicht kopiert werden kann, werden auch die noch ausstehenden Dateien und Unterordner nicht weiter kopiert. Eine Lösung dieses Problems liefert das Rezept 126.

An Stelle der systemeigenen Meldungen, die `SHFileOperation` im Fehlerfall ausgibt, wäre es zudem wesentlich schöner, wenn die Methode `CopyFolder` die Systemfehlermeldungen über das Flag `FOF_NOERRORUI` ausschalten, den Rückgabecode von `SHFileOperation` auswerten, daraus eine Fehlerbeschreibung generieren und schließlich mit dieser Beschreibung eine Ausnahme vom Typ `IOException` erzeugen würde. Leider sind aber die Rückgabewerte von `SHFileOperation` nicht dokumentiert. Die Microsoft-API-Dokumentation sagt lediglich aus, dass diese Funktion im Erfolgsfall 0 und im

Fehlerfall einen Wert ungleich 0 zurückgibt (siehe *msdn.microsoft.com/en-us/library/bb762164.aspx*). Der zurückgegebene Fehlercode lässt sich leider auch nicht über die API-Funktion `FormatMessage` in eine passende Fehlermeldung umsetzen (die damit ermittelten Fehlermeldungen sind absolut unpassend). Die Idee, stattdessen den letzten Windows-Fehlercode zu verwenden (den Sie über die `GetLastWin32Error`-Methode der Klasse `System.Runtime.InteropServices.Marshal` auslesen können), schlug bei meinen Versuchen ebenfalls mit vollkommen unpassenden Fehlermeldungen fehl. Hinzu kommt, dass umfangreiche Diskussionen in verschiedenen Newsgroups die scheinbare Unmöglichkeit, aus dem Rückgabewert von `SHFileOperation` eine Fehlerbeschreibung abzuleiten, belegen. Schade.



Abbildung 73: Automatische Fehlermeldung beim Kopieren einer Datei, die nicht geschrieben werden kann

126 Ordner mit .NET-Features ohne Abbruch bei einer Ausnahme kopieren

Mich ärgert es immer wieder, wenn Windows beim Kopieren eines umfangreichen Ordners den Kopiervorgang abbricht, nur weil einzelne Dateien gerade nicht kopiert werden können (weil diese z. B. geöffnet sind). Schöner wäre es, wenn die Kopiervorgänge beim Auftreten eines Fehlers einfach mit der nächsten Datei oder dem nächsten Unterordner fortgeführt und die jeweiligen Fehler am Ende gemeldet würden.

Die im Folgenden beschriebene Methode `CopyFolderFaultTolerant` implementiert ein solches Vorgehen. Diese Methode kopiert alle Dateien des Quellordners rekursiv ähnlich der gleichnamigen Methode des Rezepts 124. Zusätzlich wird der Anwender vor dem Überschreiben einer vorhandenen Datei gefragt, ob diese überschrieben werden soll (Abbildung 74).

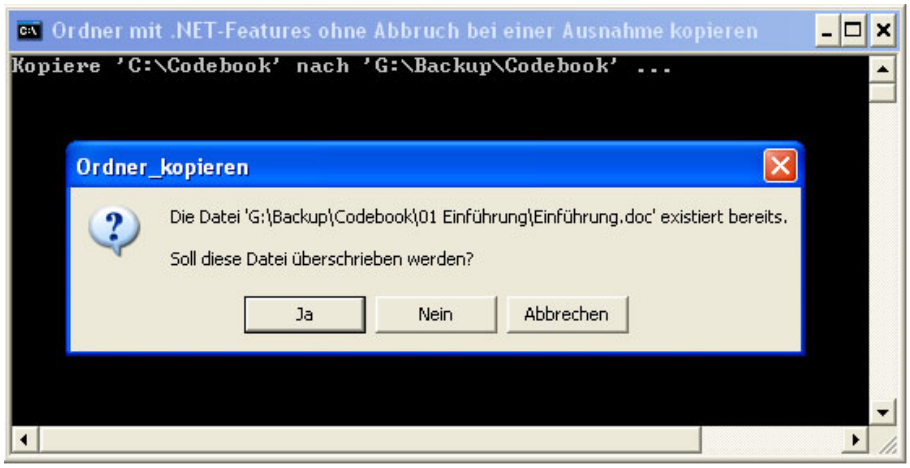


Abbildung 74: Abfrage der CopyFolderFaultTolerant-Methode zum Überschreiben einer vorhandenen Datei

Um dem Anwender die Möglichkeit zu geben, alle vorhandenen Dateien zu überschreiben, fragt die Methode einmalig ab, ob dies gewünscht ist (Abbildung 75).



Abbildung 75: Abfrage der CopyFolderFaultTolerant-Methode zum Überschreiben aller Dateien

Die dazu notwendigen Informationen werden in zwei privaten Feldern verwaltet. Beide Dialoge ermöglichen einen Abbruch, der aus Sicherheitsgründen in eine Ausnahme vom Typ `IOException` umgesetzt wird.

Fehler beim Kopieren einer Datei werden abgefangen und in einer speziellen Auflistung von Kopierfehler-Objekten protokolliert. Um dies zu realisieren habe ich zunächst eine Klasse `CopyFault` entwickelt, deren Instanzen je einen Kopierfehler speichern sollen.

Um das Programm kompilieren zu können, müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Collections.Generic`, `System.Collections.ObjectModel`, `System.IO` und `System.Windows.Forms` importieren.

```
/* Verwaltet Informationen zu fehlgeschlagenen Kopieroperationen */
public class CopyFault
{
   /* Gibt an, ob es sich um eine Datei handelt */
   public bool IsFile;

   /* Der Quellpfad */
   public string Source;

   /* Der Zielpfad */
   public string Destination;

   /* Der aufgetretene Fehler */
   public string Error;

   /* Konstruktor */
   internal CopyFault(bool isFile, string source, string destination,
      string error)
   {
      this.IsFile = isFile;
      this.Source = source;
      this.Destination = destination;
      this.Error = error;
   }
}
```

Listing 222: Klasse für die Verwaltung von Kopierfehlern

Die Klasse `FileSystemUtils` besitzt neben zwei privaten Feldern, die intern für das Überschreiben von Dateien verwendet werden, eine Methode `CopyFolderFaultTolerant`, die einen Ordner fehlertolerant kopiert. Diese Methode initialisiert zunächst die privaten Felder und ruft dann die private Methode `CopySubFoldersAndFiles` auf. `CopyFolderFaultTolerant` gibt eine Instanz auf eine `ReadOnlyCollection` mit `CopyFault`-Objekten zurück, über die später die eventuell aufgetretenen Kopierfehler ausgelesen werden können.

```
public static partial class FileSystemUtils
{
   /* Definiert, ob beim Kopieren mit CopyFolderFaultTolerant
      alle Dateien überschrieben werden sollen. */
   private static bool overwriteAllFiles;
```

Listing 223: Methode zum Start des rekursiven Kopierens eines Ordners

```
/* Gibt an, ob beim Kopieren mit CopyFolderFaultTolerant bereits gefragt
   wurde, ob alle Dateien überschrieben werden sollen. */
private static bool alreadyAskedForOverwriteAllFiles;

/* Kopiert (rekursiv) alle Unterordner und Dateien eines Ordners in
   einen anderen Ordner */
public static ReadOnlyCollection<CopyFault> CopyFolderFaultTolerant(
   string sourceFolderPath, string destFolderPath)
{
   // Liste von CopyFault-Objekten erzeugen
   List<CopyFault> copyFaults = new List<CopyFault>();

   // Datei-Überschreib-Flags voreinstellen
   FileSystemUtils.overwriteAllFiles = false;
   FileSystemUtils.alreadyAskedForOverwriteAllFiles = false;

   // Die rekursive Methode zum Kopieren der Unterordner
   // und Dateien aufrufen
   FileSystemUtils.CopySubFoldersAndFiles(
      new DirectoryInfo(sourceFolderPath),
      sourceFolderPath, destFolderPath, copyFaults);

   // Ergebnis zurückgeben
   return new ReadOnlyCollection<CopyFault>(copyFaults);
}
```

Listing 223: Methode zum Start des rekursiven Kopierens eines Ordners (Forts.)

Die private Methode `CopySubFoldersAndFiles` (Listing 224) geht zuerst alle Unterordner des Ordners durch, erzeugt den jeweiligen Unterordner im Zielordner und ruft sich danach selbst auf, um die weiteren Unterordner zu erzeugen. Danach kopiert `CopySubFoldersAndFiles` die im jeweiligen Ordner enthaltenen Dateien in den jeweiligen Zielordner.

`CopySubFoldersAndFiles` wird ein `DirectoryInfo`-Objekt übergeben, das den Ordner repräsentiert. Beim rekursiven Aufruf übergibt `CopySubFoldersAndFiles` ein solches für jeden Unterordner an sich selbst. Da innerhalb der Methode beim Kopieren der Dateien Informationen über den Haupt-Quell- und -Zielordner benötigt werden (um den zum jeweiligen Zielordner relativen Dateinamen zu ermitteln), werden dieser Methode zudem die Pfade dieser Ordner übergeben.

`CopySubFoldersAndFiles` erzeugt im Fehlerfall ein neues `CopyFault`-Objekt und hängt dieses an die Auflistung von `CopyFault`-Objekten an, die ebenfalls (über die `CopyFolderFaultTolerant`-Methode) übergeben wurde.

```
private static void CopySubFoldersAndFiles(DirectoryInfo folder,
   string mainSourceFolderPath, string mainDestFolderPath,
```

Listing 224: Methode zum rekursiven Kopieren eines Ordners und Ende der FileSystemUtils-Klasse

```
      List<CopyFault> copyFaults)
   {
      // Zielordner anlegen
      try
      {
         // Zielordnername ermitteln
         string destFolderPath = folder.FullName.Replace(mainSourceFolderPath,
            mainDestFolderPath);

         // Ordner anlegen
         Directory.CreateDirectory(destFolderPath);
      }
      catch (IOException ex)
      {
         // Fehler in der CopyFault-Auflistung dokumentieren
         copyFaults.Add(new CopyFault(false, mainSourceFolderPath,
            mainDestFolderPath, ex.Message));
      }

      // Alle Unterordner des übergebenen Ordners durchgehen
      DirectoryInfo[] subFolders = folder.GetDirectories();
      for (int i = 0; i < subFolders.Length; i++)
      {
         // Pfad für den Ziel-Unterordner ermitteln, indem der Pfad zum
         // Quellordner durch den Pfad zum Zielordner ersetzt wird
         string destSubFolderName = subFolders[i].FullName.Replace(
            mainSourceFolderPath, mainDestFolderPath);

         // Funktion rekursiv aufrufen um zunächst die weiteren Unterordner
         // zu erzeugen
         CopySubFoldersAndFiles(
            subFolders[i], mainSourceFolderPath,
            mainDestFolderPath, copyFaults);
      }

      // Die im Ordner enthaltenen Dateien ermitteln
      FileInfo[] files = folder.GetFiles();

      // Alle Dateien durchgehen
      for (int i = 0; i < files.Length; i++)
      {
         // Ziel-Dateiname ermitteln, indem der Pfad zum Quellordner
         // durch den Pfad zum Zielordner ersetzt wird
         string destFileName = files[i].FullName.Replace(
            mainSourceFolderPath, mainDestFolderPath);
```

Listing 224: Methode zum rekursiven Kopieren eines Ordners und Ende der FileSystemUtils-Klasse (Forts.)

```
// Flag setzen, das festlegt, ob die Datei kopiert werden soll
bool performCopyOperation;
performCopyOperation = true;

// Überprüfen, ob die Datei bereits existiert
if (File.Exists(destFileName))
{
   // Fragen, ob die Datei überschrieben werden soll, falls der
   // Anwender dies zuvor noch nicht für alle Dateien und Ordner
   // gemeinsam bestätigt hat
   if (FileSystemUtils.overwriteAllFiles == false)
   {
      switch (MessageBox.Show("Die Datei '" + destFileName +
         "' existiert bereits.\r\n\r\n" +
         "Soll diese Datei überschrieben werden?",
         Application.ProductName, MessageBoxButtons.YesNoCancel,
         MessageBoxIcon.Question))
      {
         case DialogResult.Yes:
            // Nachfragen, ob alle Dateien überschrieben werden
            // sollen, falls dies noch nicht geschehen ist
            if (FileSystemUtils.alreadyAskedForOverwriteAllFiles ==
               false)
            {
               switch (MessageBox.Show("Sollen alle im Zielordner " +
                  "vorhandenen Dateien überschrieben werden?",
                  Application.ProductName,
                  MessageBoxButtons.YesNoCancel,
                  MessageBoxIcon.Question))
               {
                  case DialogResult.Yes:
                     // Flag setzen, das das Überschreiben steuert
                     FileSystemUtils.overwriteAllFiles = true;
                     break;

                  case DialogResult.Cancel:
                     // Anwender hat abgebrochen: Ausnahme erzeugen,
                     // da der Ordner nicht komplett kopiert werden
                     // konnte
                     throw new IOException("Benutzerabbruch");
               }

               // Festlegen, dass nicht mehr nachgefragt werden muss
               FileSystemUtils.alreadyAskedForOverwriteAllFiles =
                  true;
```

Listing 224: Methode zum rekursiven Kopieren eines Ordners und Ende der FileSystemUtils-Klasse (Forts.)

```
                    }
                    break;

                case DialogResult.No:
                    // Festlegen, dass die aktuelle Datei nicht kopiert
                    // werden soll
                    performCopyOperation = false;
                    break;

                case DialogResult.Cancel:
                    // Anwender hat abgebrochen: Ausnahme erzeugen, da der
                    // Ordner nicht komplett kopiert werden konnte
                    throw new IOException("Benutzerabbruch");
            }
        }
    }

    // Datei kopieren, wenn die Operation ausgeführt werden soll
    if (performCopyOperation)
    {
        try
        {
            File.Copy(files[i].FullName, destFileName,
                true);
        }
        catch (Exception ex)
        {
            // Fehler in der CopyFault-Auflistung dokumentieren
            copyFaults.Add(new CopyFault(true, files[i].FullName,
                destFileName, ex.Message));
        }
    }
    else
    {
        // Datei sollte nicht überschrieben werden: Fehler in der
        // CopyFault-Auflistung dokumentieren
        copyFaults.Add(new CopyFault(true, files[i].FullName,
            destFileName,
            "Fehlende Anwender-Erlaubnis zum Überschreiben"));
    }
  }
}
}
```

Listing 224: Methode zum rekursiven Kopieren eines Ordners und Ende der FileSystemUtils-Klasse (Forts.)

Bei der Anwendung der `CopyFolderFaultTolerant`-Methode können Sie die zurückgegebene `ReadOnlyCollection<CopyFault>`-Auflistung dazu verwenden, eventuelle Kopierfehler zu ermitteln. Damit können Sie die nicht kopierbaren Dateien auflisten, in eine Textdatei schreiben oder dem Anwender zum erneuten Kopieren zur Verfügung stellen. Der folgende Beispielcode listet die Fehler lediglich auf:

```
string sourceFolderName = @"C:\Codebook";
string destFolderName = @"C:\Backup\Codebook";

Console.WriteLine("Kopiere '" + sourceFolderName + "' nach '" +
   destFolderName + "' ...");

// Ordner kopieren
try
{
   ReadOnlyCollection<CopyFault> copyFaults =
      FileSystemUtils.CopyFolderFaultTolerant(sourceFolderName,
      destFolderName);
   if (copyFaults.Count == 0)
   {
      Console.WriteLine("Fertig");
   }
   else
   {
      // Beim Kopieren sind Fehler aufgetreten: Alle Fehler
      // durchgehen und ausgeben
      foreach (CopyFault copyFault in copyFaults)
      {
         Console.WriteLine();
         if (copyFault.IsFile)
         {
            // Es handelt sich um eine Datei
            Console.WriteLine("Fehler beim Kopieren der " +
               "Datei '{0}' nach '{1}': {2}",
               copyFault.Source, copyFault.Destination,
               copyFault.Error);
         }
         else
         {
            // Es handelt sich um einen Ordner
            Console.WriteLine("Fehler beim Kopieren des " +
               "Ordners '{0}' nach '{1}': {2}",
               copyFault.Source, copyFault.Destination,
               copyFault.Error);
         }
```

Listing 225: Anwendung der Klasse zum komfortablen Kopieren

```
      }
   }
}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Kopieren des Ordners: {0}",
      ex.Message);
}
```

Listing 225: Anwendung der Klasse zum komfortablen Kopieren (Forts.)

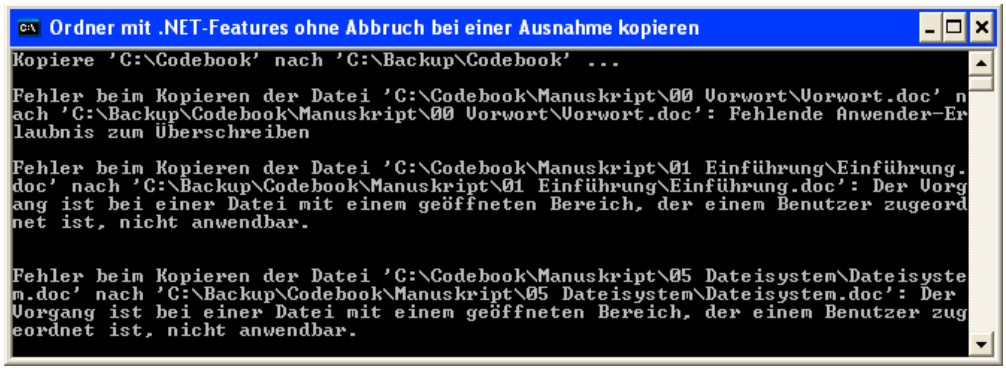


Abbildung 76: Das Programm wurde ausgeführt und konnte einige Dateien nicht kopieren

127 Ordner umbenennen

Ich hatte, wie viele andere auch, ursprünglich vermutet, dass zum Umbenennen eines Ordners eine *Rename*-Methode existieren würde. Dies ist jedoch nicht der Fall. Einen Ordner können Sie aber über die `Move`-Methode der `Directory`-Klasse (aus dem Namensraum `System.IO`) umbenennen. `Move` erzeugt eine Ausnahme vom Typ `IOException`, wenn das Umbenennen nicht möglich ist. Das folgende Beispiel beachtet diese Tatsache:

```
string sourceDirName = @"C:\Temp\DemoFolder";  // Alter Name
string destDirName = @"C:\Temp\Demo Folder";   // Neuer Name

try
{
   // Ordner umbenennen
   Directory.Move(sourceDirName, destDirName);

   Console.WriteLine("Ordner erfolgreich umbenannt");
}
catch (IOException ex)
```

Listing 226: Umbenennen eines Ordners ohne Undo-Option

```
{
   Console.WriteLine("Fehler beim Umbenennen des Ordners: " + ex.Message);
}
```

Listing 226: Umbenennen eines Ordners ohne Undo-Option (Forts.)

128 Ordner verschieben

Wenn Sie einen Ordner verschieben wollen, verwenden Sie prinzipiell denselben Programmcode wie in Rezept 127. Sie geben der `Move`-Methode als Zielordner lediglich einen Ordner an, der einen anderen übergeordneten Ordner oder ein anderes Laufwerk aufweist.

Das Beispiel in Listing 227 erfordert den Import der Namensräume `System` und `System.IO`.

```
string sourceFolderName = @"C:\DemoFolder";
string destFolderName = @"C:\Temp\DemoFolder";

try
{
   // Ordner verschieben
   Directory.Move(sourceFolderName, destFolderName);

   Console.WriteLine("Ordner erfolgreich verschoben");
}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Verschieben des Ordners: " +
      ex.Message);
}
```

Listing 227: Verschieben eines Ordners ohne Undo-Option

Hinweis

Das Verschieben von Ordnern über die `Move`-Methode ist nicht allzu komfortabel. Problematisch ist z.B., dass diese Methode mit einer Ausnahme abbricht, wenn eine Zieldatei oder ein Zielordner bereits existiert. Im Rezept 148 finden Sie eine Methode, die das erweiterte Umbenennen und Verschieben von Dateien oder Ordnern über die API-Funktion `SHFileOperation` implementiert. Diese Methode zeigt einen automatischen Fortschrittsdialog an, ermöglicht den Abbruch der Aktion und das Überschreiben vorhandener Dateien bzw. Ordner mit oder ohne Nachfrage beim Anwender.

129 Ordner löschen

Wenn Sie einen Ordner löschen wollen, können Sie dazu die Methode `Delete` der Klasse `System.IO.Directory` verwenden. Am ersten Argument übergeben Sie den Pfadnamen des Ordners, im optionalen zweiten Argument können Sie mit `true` bestimmen, dass auch alle Unterordner und enthaltenen Dateien gelöscht werden:

```
string folderName = @"C:\Test\Test\Test";
try
{
   // Ordner löschen
   Directory.Delete(folderName, true);
}
catch (Exception ex)
{
   MessageBox.Show(
      "Fehler beim Löschen des Ordners '" + folderName + "': " +
      ex.Message, Application.ProductName, MessageBoxButtons.OK,
      MessageBoxIcon.Error);
}
```

Listing 228: Löschen eines Ordners

Dieses Beispiel benötigt aufgrund der Verwendung der `MessageBox`-Klasse die Referenzierung der Assembly *System.Windows.Forms.dll* und die Einbindung des Namensraums `System.Windows.Forms`. Daneben müssen die Namensräume `System` und `System.IO` importiert werden.

Übergeben Sie am zweiten Argument nichts oder `false`, führt das Löschen eines nicht leeren Ordners zu einer Ausnahme vom Typ `IOException`. Der zu löschende Ordner muss außerdem existieren und löschbar sein (nicht schreibgeschützt), ansonsten erzeugt `Delete` ebenfalls eine Ausnahme. Dieselbe Ausnahme wird natürlich auch erzeugt, wenn die Pfadangabe syntaktisch nicht korrekt ist.

Hinweis

Beachten Sie, dass `Delete` Ordner und alle enthaltenen Dateien dauerhaft löscht und nicht in den Papierkorb verschiebt. Das Verschieben in den Papierkorb finden Sie im Rezept 149.

130 Ordnergröße ermitteln

Die für allgemeine Verzeichnisinformationen zuständige Klasse `DirectoryInfo` bietet keine direkte Möglichkeit, die gesamte Größe eines Ordners abzufragen. Leider besteht auch scheinbar keine Möglichkeit, die Ordnergröße über Windows-API-Funktionen oder über WMI abzufragen. Um die Summe der Dateigrößen aller in einem Ordner und dessen Unterordnern enthaltenen Dateien zu ermitteln, müssen Sie den Ordner rekursiv durchgehen, die Dateigröße jeder enthaltenen Datei ermitteln und summieren.

Dazu können Sie ein `DirectoryInfo`-Objekt verwenden. Über dessen `GetDirectories`-Methode erhalten Sie ein `DirectoryInfo`-Array für alle Unterordner. Damit können Sie einen Ordner rekursiv durchgehen. Die `GetFiles`-Methode eines solchen Objekts gibt Ihnen Zugriff auf die enthaltenen Dateien in Form eines Arrays von `FileInfo`-Objekten, aus dessen Eigenschaft `Size` Sie die Größe auslesen können.

Die Methode `GetFolderSize` in Listing 229 setzt das rekursive Durchgehen eines Ordners und das Summieren der Dateigrößen um. `GetFolderSize` geht zuerst alle Dateien des übergebenen

Ordners durch um deren Größe in der Variablen `folderSize` zu summieren. Danach geht diese Methode alle Unterordner durch und ruft sich selbst für jeden dieser Ordner auf. `GetFolderSize` gibt die jeweils ermittelte Ordnergröße zurück. Deshalb kann die Variable `folderSize` um den jeweiligen Rückgabewert erhöht werden. Der erste Aufruf in der rekursiven Kette gibt schließlich die ermittelte Größe an die Methode `GetFolderSize` zurück. Einer zweiten Variante dieser Methode wird ein String übergeben, der den Pfad zum Ordner angeht. Diese Variante ruft einfach die erste Variante mit einem neu erzeugten `DirectoryInfo`-Objekt auf.

Um die Methoden aus Listing 229 kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
public static long GetFolderSize(DirectoryInfo folder)
{
   // Variable zur Speicherung der jeweils ermittelten Dateigrößen-Summe
   // eines (Unter-)Ordners
   long folderSize = 0;

   // Summe der Größen der Dateien im Ordner ermitteln
   FileInfo[] files = folder.GetFiles();
   for (int i = 0; i < files.Length; i++)
   {
      folderSize += files[i].Length;
   }

   // Unterordner ermitteln und die Methode rekursiv für jeden Unterordner
   // aufrufen und dabei die Variable folderSize hochzählen
   DirectoryInfo[] subFolders = folder.GetDirectories();
   for (int i = 0; i < subFolders.Length; i++)
   {
      folderSize += GetFolderSize(subFolders[i]);
   }

   // Die ermittelte Ordnergröße zurückgeben
   return folderSize;
}

public static long GetFolderSize(string folderPath)
{
   // Die rekursive Methode zur Ermittlung der Größe eines Ordners aufrufen
   return GetFolderSize(new DirectoryInfo(folderPath));
}
```

Listing 229: Methoden zum rekursiven Ermitteln der Größe eines Ordners

Abbildung 77 zeigt eine Anwendung, die die Größe des Windows-Systemordners über `GetFolderSize` ermittelt hat.

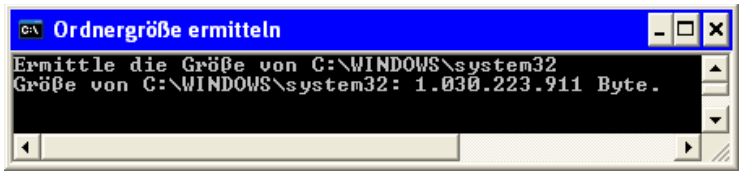


Abbildung 77: Beispielprogramm zur Ermittlung der Größe des Windows-Systemordners

Tipp Den ermittelten Bytewert können Sie über das Rezept 15 in ein besser lesbares Format mit der Einheit Byte, KB, MB oder GB umwandeln.

131 Ordnergröße aller direkten Unterordner ermitteln

Basierend auf dem Rezept 130 ist es ein Leichtes, die Größe aller direkten Unterordner eines Ordners zu ermitteln. So können Sie z. B. feststellen, welche Ordner besonders viele und/oder große Dateien speichern.

Als Lösung dieses »Problems« habe ich eine Methode `GetSubFolderSizes` entwickelt. Diese Methode verwendet ein `DirectoryInfo`-Objekt zum Durchgehen der Unterordner eines anzugebenden Ordners und ruft für jeden dieser Ordner die Methode `GetFolderSize` aus dem vorhergehenden Rezept auf. Um dem Programmierer zu ermöglichen, dem Anwender den Fortschritt der bei großen Ordnern recht langwierigen Operation anzuzeigen, ruft `GetSubFolderSizes` für jeden neuen Ordner einen Delegate auf, der auf eine Methode verweist, der der Name des aktuellen Ordners übergeben wird.

`GetSubFolderSizes` gibt eine `ReadOnlyCollection<T>`-Auflistung von speziellen `FolderSize`-Objekten zurück.

Zum Kompilieren der im Folgenden beschriebenen Klassen müssen Sie die Namensräume `System.Collections.Generic`, `System.Collections.ObjectModel` und `System.IO` importieren.

Für die Implementierung muss zunächst der Delegate deklariert werden:

```
public delegate void FolderProgressHandler(string currentFolderPath);
```

Listing 230: Deklaration eines Delegate für den Fortschritt des Auslesens der Ordnergrößen

Dann folgt die Klasse, deren Instanzen in der Auflistung gespeichert werden sollen, die von `GetSubFolderSizes` zurückgegeben wird:

```
public class FolderSize
{
   /* Der Pfad zum Ordner */
   public string FolderPath;
```

Listing 231: Klasse für die Speicherung von Ordnergrößen

```
   /* Die Gesamtgröße aller im Ordner und dessen Unterordnern
      enthaltenen Dateien */
   public long Size;

   /* Konstruktor */
   internal FolderSize(string folderPath, long size)
   {
      this.FolderPath = folderPath;
      this.Size = size;
   }
}
```

Listing 231: Klasse für die Speicherung von Ordnergrößen (Forts.)

Der Methode `GetSubFolderSizes` in Listing 32 wird neben dem Namen des Ordners eine Instanz des Fortschritt-Delegates (oder `null`) übergeben. Der Aufrufer muss dann eine dem Delegate entsprechende Methode zur Verfügung stellen, mit dieser eine neue Instanz des Delegates erzeugen und diese dann an `GetSubFolderSizes` übergeben. Alternativ kann der Aufrufer auch einfach `null` übergeben, wenn er die Fortschrittsmeldung nicht wünscht.

`GetSubFolderSizes` macht dann nicht mehr allzu viel. Zunächst werden die Ergebnis-Auflistung und ein `DirectoryInfo`-Objekt für den Ordner erzeugt. Dann ruft diese Methode den Fortschritts-Delegate zunächst für den übergeordneten Ordner auf, ermittelt die Gesamtgröße der in diesem Ordner gespeicherten Dateien und geht schließlich alle Unterordner durch, wobei deren Größe über die Methode `GetFolderSize` aus dem Rezept 130 ermittelt wird. Für jeden Unterordner wird natürlich wieder der Fortschritts-Delegate aufgerufen. Alle ermittelten Ordnergrößen werden in einer lokalen `FolderSize`-Auflistung gespeichert, die schließlich von `GetSubFolderSizes` als `ReadOnlyCollection<FolderSize>` zurückgegeben wird.

```
public static ReadOnlyCollection<FolderSize> GetSubFolderSizes(
   string folderPath, FolderProgressHandler progress)
{
   // FolderSizes-Objekt für die Speicherung der Ordnergrößen erzeugen
   List<FolderSize> folderSizes = new List<FolderSize>();

   // DirectoryInfo-Objekt für den Ordner erzeugen
   DirectoryInfo folder = new DirectoryInfo(folderPath);

   // Fortschritts-Delegate aufrufen
   if (progress != null)
   {
      progress(folder.Name);
   }
```

Listing 232: Methode zum Ermitteln der Größe aller direkten Unterordner eines Ordners

```
   // Gesamtgröße der Dateien des Ordners ermitteln
   FileInfo[] files = folder.GetFiles();
   long folderSize = 0;
   for (int i = 0; i < files.Length; i++)
   {
      folderSize += files[i].Length;
   }

   // Ordnername und -größe als erstes Element in der Auflistung ablegen
   folderSizes.Add(new FolderSize(folderPath, folderSize));

   // Unterordner ermitteln
   DirectoryInfo[] subFolders = folder.GetDirectories();

   // Unterordner durchgehen
   for (int i = 0; i < subFolders.Length; i++)
   {
      // Fortschritts-Delegate aufrufen
      if (progress != null)
      {
         progress(subFolders[i].Name);
      }

      // Die jeweilige Größe ermitteln
      folderSize = GetFolderSize(subFolders[i]);

      // Ordnername und -größe in der Auflistung ablegen
      folderSizes.Add(new FolderSize(subFolders[i].FullName, folderSize));
   }

   // Ergebnis-Auflistung zurückgeben
   return new ReadOnlyCollection<FolderSize>(folderSizes);
}
```

Listing 232: Methode zum Ermitteln der Größe aller direkten Unterordner eines Ordners (Forts.)

Listing 233 zeigt eine Beispielanwendung der `GetSubFolderSizes`-Methode. Diese Konsolenanwendung ermittelt die Größen der Unterordner con *C:\Inetpub*. Der Fortschritt wird über eine anonyme Methode ausgegeben, die als Lambda-Ausdruck übergeben wird. Das Programm benötigt den Import der Namensräume `System` und `System.IO`.

```
// Ordner festlegen
string folderName = "C:\\Inetpub";

try
{
   // Rekursives Ermitteln der Größe der im Ordner direkt und in allen
   // Unterordnern gespeicherten Dateien. Übergeben werden der Ordnername
   // und eine neue Instanz des Fortschritts-Delegate
   ReadOnlyCollection<FolderSize> folderSizes =
      GetSubFolderSizes(folderName,
      searchFolderName => Console.WriteLine(
      "Durchsuche {0} ...", searchFolderName));

   // Die Auflistung der Ordnergrößen durchgehen
   Console.WriteLine("\r\nErmittelte Größen:");
   for (int i = 0; i < folderSizes.Count; i++)
   {
      Console.WriteLine("{0}: {1:#,#0} Byte",
         folderSizes[i].FolderPath, folderSizes[i].Size);
   }
}
catch (IOException ex)
{
   Console.WriteLine("Fehler: " + ex.Message);
}
```

Listing 233: Anwendung der Methode zum Ermitteln der Größen aller Unterordner eines Ordners

Das Ergebnis dieses Programms zeigt Abbildung 78.

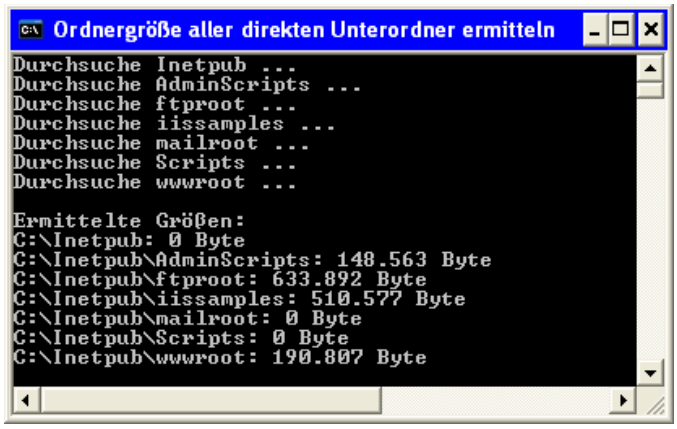


Abbildung 78: Das Beispielprogramm zur Ermittlung von Ordnergrößen in Aktion

132 Überprüfen, ob eine Datei existiert

Wenn Sie überprüfen wollen, ob eine Datei existiert, können Sie ähnlich der Überprüfung der Existenz eines Ordners (Rezept 122) wieder zwischen zwei Varianten wählen. Zum einen können Sie die statische Methode `Exists` der `File`-Klasse (aus dem Namensraum `System.IO`) einsetzen:

```
string fileName = "c:\\autoexec.bat";
if (File.Exists(fileName))
{
   Console.WriteLine(fileName + " existiert");
}
else
{
   Console.WriteLine(fileName + " existiert nicht");
}
```

Die zweite Variante setzt eine Instanz der Klasse `FileInfo` (ebenfalls aus dem Namensraum `System.IO`) ein und fragt deren `Exists`-Methode ab:

```
string fileName = "c:\\autoexec.bat";
FileInfo di = new FileInfo(fileName);
if (di.Exists)
{
   Console.WriteLine(fileName + " existiert");
}
else
{
   Console.WriteLine(fileName + " existiert nicht");
}
```

Beide Varianten können Sie auch ohne Probleme im Netzwerk einsetzen. Voraussetzung dafür ist natürlich, dass der aktuell in Windows eingeloggte Benutzer Leserechte für die Netzwerk-Ressource besitzt.

133 Dateien in einem Ordner und seinen Unterordnern suchen

In einem bestimmten Ordner können Sie Dateien über die `GetFiles`-Methode einer `DirectoryInfo`-Instanz suchen. Dieser Methode übergeben Sie entweder nichts oder ein Dateimuster (wie z. B. *.txt). Wenn Sie nichts übergeben, sucht diese Methode alle Dateien. Übergeben Sie ein Dateimuster, sucht die Methode nur die Dateien, die dem Muster entsprechen. Dummerweise erlaubt das Muster keine mehrfachen Angaben. So können Sie z. B. nicht nach allen Bilddateien suchen, die die Endungen *.jpg*, *.jpeg*, *.gif*, oder *.bmp* aufweisen. Außerdem sucht `GetFiles` nur in dem angegebenen Ordner und nicht in dessen Unterordnern.

Eine Suche, bei der Sie mehrere Dateimuster angeben können und die auch in allen Unterordnern sucht, müssen Sie selbst programmieren. Meine Lösung dazu finden Sie in Listing 234. Die öffentliche Methode `FindFiles` erzeugt zunächst eine generische `List`-Auflistung mit dem Typ `FileInfo`. Diese Auflistung ist die Basis für die von `FindFiles` zurückgegebene `ReadOnlyCollection`. `FindFiles` ruft dann die private Methode `findFiles` auf und übergibt neben der erzeugten Auflistung und den weiteren Argumenten eine neues `DirectoryInfo`-Objekt, das den Startordner repräsentiert. `findFiles` ermittelt zunächst alle Dateien in dem übergebenen Ord-

ner, die dem angegebenen Suchmuster entsprechen. Um mehrere Suchmuster zu ermöglichen geht diese Methode davon aus, dass diese durch Semikolons voneinander getrennt angegeben werden (wie es allgemein üblich ist), und splittet das übergebene Suchmuster-Argument entsprechend, wenn es nicht leer ist. Dann liest sie für jedes einzelne der ermittelten Muster die dazu passenden Dateien ein und speichert diese in der übergebenen Datei-Auflistung `fileList`. Bei einem leeren Muster werden einfach `FileInfo`-Objekte für alle Dateien im angegebenen Ordner in die Datei-Auflistung geschrieben. Um auch in allen Unterordnern zu suchen geht `findFiles` diese schließlich durch, wenn das Argument `recursive` `true` ist, und ruft sich selbst mit dem jeweiligen Unterordner als neuem Startordner auf. Auf diese Weise werden der ursprünglich übergebene und alle Unterordner durchsucht.

Die beiden Such-Methoden in Listing 234 erfordern den Import der Namensräume `System.Collections.Generic`, `System.Collections.ObjectModel` und `System.IO`.

```
public static ReadOnlyCollection<FileInfo> FindFiles(
   string startDirectory, string filePattern, bool recursive)
{
   // Basis-Auflistung erzeugen
   List<FileInfo> fileList = new List<FileInfo>();

   // Die rekursive private Methode aufrufen
   findFiles(new DirectoryInfo(startDirectory), filePattern,
     recursive, fileList);

   // Ergebnis-Collection zurückgeben
   return new ReadOnlyCollection<FileInfo>(fileList);
}

private static void findFiles(DirectoryInfo directory,
   string filePattern, bool recursive, List<FileInfo> fileList)
{
   if (filePattern != null && filePattern.Length > 0)
   {
      // Das Dateimuster splitten
      string[] filePatterns = filePattern.Split(';');

      // Alle Dateimuster durchgehen und in dem übergebenen Verzeichnis
      // suchen
      foreach (string partPattern in filePatterns)
      {
         foreach (FileInfo fileInfo in directory.GetFiles(partPattern))
         {
            fileList.Add(fileInfo);
         }
      }
```

Listing 234: Öffentliche und private Methode zum rekursiven Suchen aller Dateien, die einem angegebenen Muster entsprechen

```
   }
   else
   {
         // Kein Suchmuster angegeben: Alle Dateien durchgehen
         foreach (FileInfo fileInfo in directory.GetFiles())
         {
            fileList.Add(fileInfo);
         }
   }

   if (recursive)
   {
      // Wenn rekursiv gesucht werden soll:
      // Die Methode für alle Unterordner aufrufen
      foreach (DirectoryInfo subDirectory in directory.GetDirectories())
      {
         findFiles(subDirectory, filePattern, recursive, fileList);
      }
   }
}
```

Listing 234: Öffentliche und private Methode zum rekursiven Suchen aller Dateien, die einem angegebenen Muster entsprechen (Forts.)

Listing 235 zeigt eine beispielhafte Anwendung der `FindFiles`-Methode an der Konsole.

```
// Alle Bilddateien im Windows-Ordner suchen
string startDirectory = "C:\\Windows";
string filePattern = "*.jpg;*.jpeg;*.gif;*.bmp;*.tif;*.tiff";
ReadOnlyCollection<FileInfo> fileList =
   FindFiles(startDirectory, filePattern, true);

// Die gefundenen Dateien durchgehen und ausgeben
foreach (FileInfo fi in fileList)
{
   Console.WriteLine(fi.FullName);
}
```

Listing 235: Beispielhafte Anwendung der FindFiles-Methode: Suchen verschiedener Bilddateien im Windows-Ordner

134 Programmdateien in den Systempfaden suchen

Programme, die im Windows-, im Systemordner oder in einem der Ordner gespeichert sind, die in der Umgebungsvariable `Path` eingetragen sind, können Sie zum Starten üblicherweise ohne Pfad angeben. Windows sucht automatisch in diesen Ordnern nach der Programmdatei.

Wenn Sie ermitteln wollen, ob ein zu startendes Programm in einem der Systempfad-Ordner gespeichert ist, können Sie die folgende Methode `FindFileInSystemPaths` einsetzen, die eine Datei im System-, im Windows- und in den in `Path` angegebenen Ordnern sucht.

Der Windows-Ordner wird dabei über die API-Funktion `GetWindowsDirectory` ermittelt, da das .NET Framework dazu keine direkte Möglichkeit bietet. Der Inhalt der `Path`-Variable wird über die Methode `GetEnvironmentVariable` der Klasse `System.Environment` ausgelesen. Da Pfade im Systempfad auch in Anführungszeichen eingebettet sein können, berücksichtigt `FindFileInSystemPaths` diesen Umstand und entfernt eventuelle Anführungszeichen. `FindFileInSystemPaths` sucht die angegebene Datei in der Reihenfolge, die auch Windows einsetzt (Systemordner – Windows-Ordner – `Path`-Pfade).

`FindFileInSystemPaths` erfordert den Import der Namensräume `System`, `System.Text`, `System.IO` und `System.Runtime.InteropServices`.

```
/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll", SetLastError = true)]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
   uint uSize);

/* Sucht eine Datei in den Systempfaden */
public static string FindFileInSystemPaths(string fileName)
{
   string path = null;

   // Im Windows-Systemordner suchen
   path = Path.Combine(Environment.SystemDirectory, fileName);
   if (File.Exists(path))
   {
      return path;
   }

   // Im Windows-Ordner suchen
   const int MAX_PATH = 160;
   string windowsDirectoryName = null;
   StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
   if (GetWindowsDirectory(buffer, 260) > 0)
   {
      windowsDirectoryName = buffer.ToString();
      path = Path.Combine(windowsDirectoryName, fileName);
      if (File.Exists(path))
      {
         return path;
      }
   }
```

Listing 236: Methode zum Suchen von Dateien in den Systempfaden

```
   // In den Ordnern suchen, die in der Umgebungsvariablen Path eingestellt
   // sind
   string[] systemPaths = Environment.GetEnvironmentVariable(
      "path").Split(new char[] { ';' });
   for (int i = 0; i < systemPaths.Length; i++)
   {
      path = Path.Combine(systemPaths[i].Trim().Trim('\"'), fileName);
      if (File.Exists(path))
      {
         return path;
      }
   }

   return null;
}
```

Listing 236: Methode zum Suchen von Dateien in den Systempfaden (Forts.)

Listing 237 zeigt eine Beispielanwendung der `FindFileInSystemPaths`-Methode.

```
// Suchen der Datei notepad.exe
string fileName = FindFileInSystemPaths("notepad.exe");
if (fileName != null)
{
   Console.WriteLine("Gefunden: {0}", fileName);
}
else
{
   Console.WriteLine("Nicht gefunden");
}

// Suchen der Datei java.exe
fileName = FindFileInSystemPaths("java.exe");
if (fileName != null)
{
   Console.WriteLine("Gefunden: {0}", fileName);
}
else
{
   Console.WriteLine("Nicht gefunden");
}
```

Listing 237: Suchen der Dateien Notepad.exe und Java.exe in den Systempfaden

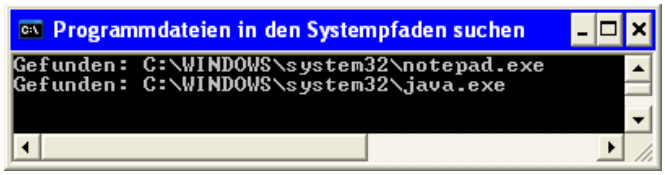


Abbildung 79: Das Beispielprogramm hat die Dateien gefunden

135 Dateien kopieren

Dateien können Sie sehr einfach über die `Copy`-Methode der `File`-Klasse kopieren. Am ersten Argument übergeben Sie dieser Methode den vollen Namen der Quelldatei, am zweiten den der Zieldatei. Über das dritte Argument können Sie festlegen, ob eine vorhandene Datei überschrieben werden soll. Tragen Sie hier `true` ein, wird eine eventuell vorhandene Datei ohne weitere Warnung überschrieben.

Tritt beim Kopieren ein Fehler auf, z. B. weil die Quelldatei gerade geöffnet oder die Zieldatei vorhanden und schreibgeschützt ist, erzeugt `Copy` eine Ausnahme vom Typ `IOException`, die Sie abfangen sollten.

Listing 238 zeigt das Kopieren einer Datei am Beispiel. Aufgrund der Verwendung der `MessageBox`-Klasse erfordert dieses Beispiel die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import des Namensraums `System.Windows.Forms`. Daneben müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
/* Variablen für die Argumente der Copy-Methode */
string sourceFileName = @"C:\boot.ini";
string destFileName = @"C:\boot.ini.copy";
bool overwrite = true;

try
{
   /* Datei kopieren */
   File.Copy(sourceFileName, destFileName, overwrite);
}
catch (IOException ex)
{
   MessageBox.Show("Die Datei kann nicht kopiert werden: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK,
      MessageBoxIcon.Exclamation);
}
```

Listing 238: Einfaches Kopieren einer Datei

Wenn Sie am dritten Argument der `Copy`-Methode `false` eintragen, erzeugt diese Methode eine Ausnahme vom Typ `IOException`, falls eine Datei mit identischem Namen bereits existiert. Da diese Ausnahme auch in anderen Fällen generiert wird, können Sie dem Anwender für den Fall, dass eine Datei gleichen Namens bereits vorhanden ist, leider nur eine allgemeine Mel-

dung übergeben. Um etwas flexibler zu sein und den Anwender zu fragen, ob eine vorhandene Datei überschrieben werden soll, können Sie vor dem Kopieren überprüfen, ob die Datei existiert. Die Funktion `CopyFile` in Listing 239 übernimmt diese Aufgabe und wertet daneben auch eventuelle Ausnahmen aus.

Die Abfrage nach einer eventuell existierenden Datei gleichen Namens erfolgt über die `Exists`-Methode der `File`-Klasse. Im positiven Fall fragt `CopyFile` den Anwender über die `MessageBox`-Klasse, ob er die Datei überschreiben will. Wählt der Anwender den Nein-Schalter (was in der Rückgabe von `DialogResult.No` resultiert), wird die Funktion mit der Rückgabe von `false` beendet. Ansonsten kopiert `CopyFile` die Datei über die `Copy`-Methode der `File`-Klasse, wobei am dritten Argument `true` übergeben wird, damit eine eventuell vorhandene Datei gleichen Namens überschrieben wird.

Konnte die Datei kopiert werden, liefert `CopyFile` `true` zurück, ansonsten `false`.

Wie bereits beim Programm in Listing 238 müssen Sie zum Kompilieren der `CopyFile`-Methode die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System.Windows.Forms` und `System.IO` importieren.

```
public static bool CopyFile(string sourceFilePath, string destFilePath)
{
   // Abfragen, ob die Datei existiert
   if (System.IO.File.Exists(destFilePath))
   {
      // Den Anwender fragen, ob die Datei überschrieben werden soll
      if (MessageBox.Show("Die Datei '" + destFilePath +
         "' existiert bereits.\r\n\r\nSoll diese Datei " +
         "überschrieben werden?",
         Application.ProductName, MessageBoxButtons.YesNo,
         MessageBoxIcon.Exclamation) == DialogResult.No)

         // Funktion beenden und false zurückgeben
         return false;
   }

   try
   {
      // Datei kopieren
      System.IO.File.Copy(sourceFilePath, destFilePath, true);

      return true;
   }
   catch (Exception ex)
   {
```

Listing 239: Erweiterte Methode zum Kopieren einer Datei

```
      // Fehlermeldung ausgeben
      MessageBox.Show("Die Datei '" + sourceFilePath + "' kann nicht " +
         "kopiert werden: " + ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

      return false;
   }
}
```

Listing 239: Erweiterte Methode zum Kopieren einer Datei (Forts.)

136 Dateien umbenennen

Dateien können Sie sehr einfach über die `Move`-Methode der `File`-Klasse aus dem Namensraum `System.IO` umbenennen. Diese Methode generiert eine Ausnahme vom Typ `IOException`, wenn das Umbenennen nicht möglich ist. Der folgende Programmcode, der den Import der Namensräume `System` und `System.IO` voraussetzt, nennt eine Datei *c:\autoexec.bat.copy* in *c:\autoexec.bat.bak* um und fängt mögliche Ausnahmen ab:

```
string sourceFileName = @"c:\autoexec.bat.copy";
string destFileName = @"c:\autoexec.bat.bak";

try
{
   // Datei umbenennen
   File.Move(sourceFileName, destFileName);

   Console.WriteLine("Datei erfolgreich umbenannt");

}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Umbenennen der Datei: " + ex.Message);
}
```

Listing 240: Umbenennen einer Datei

137 Dateien verschieben

Das Verschieben einer Datei entspricht dem Umbenennen aus Rezept 136, nur dass Sie hier einen anderen Zielordner angeben. Das Programm benötigt den Import der Namensräume `System` und `System.IO`.

```
string sourceFileName = @"C:\autoexec.bat.copy";
string destFileName = @"C:\Temp\autoexec.bat.copy";

try
{
   // Datei verschieben
   File.Move(sourceFileName, destFileName);

   Console.WriteLine("Datei erfolgreich verschoben");

}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Verschieben der Datei: " +
     ex.Message);
}
```

Listing 241: Verschieben einer Datei

138 Dateien löschen

Dateien können Sie über die Methode `Delete` der Klasse `System.IO.File` dauerhaft löschen:

```
string fileName = @"C:\boot.ini.copy";
System.IO.File.Delete(fileName);
```

Da beim Löschen Fehler auftreten können, die in einer Ausnahme vom Typ `IOException` resultieren, sollten Sie diese abfangen. Die Methode `DeleteFile` in Listing 242 setzt dies um. Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Windows.Forms` und `System.IO` importieren.

```
public static bool DeleteFile(string fileName)
{
   try
   {
      // Datei löschen
      System.IO.File.Delete(fileName);
   }
   catch (Exception ex)
   {
      // Fehlermeldung ausgeben
      MessageBox.Show("Die Datei '" + fileName + "' kann nicht " +
         "gelöscht werden: " + ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
```

Listing 242: Methode zum Löschen einer Datei mit Fehlermeldung

```
      // Misserfolg zurückmelden
      return false;
   }

   // Erfolg zurückmelden
   return true;
}
```

Listing 242: Methode zum Löschen einer Datei mit Fehlermeldung (Forts.)

Hinweis

So gelöschte Dateien werden leider nicht in den Papierkorb verschoben, sondern dauerhaft gelöscht. Eine Lösung zum Verschieben von Dateien und Ordnern in den Papierkorb finden Sie im Rezept 149.

139 Dateien vergleichen

Für den Vergleich zweier Dateien bietet das .NET Framework keinen Support. Wenn Sie zwei Dateien daraufhin vergleichen wollen, ob diese identisch sind, müssen Sie eine eigene Vergleichsmethode programmieren. Sie können natürlich auch einfach die Methode `CompareFiles` aus Listing 243 verwenden ☺. Diese Methode erwartet neben den Dateinamen der zu vergleichenden Dateien am dritten Argument einen Wert der in Listing 243 deklarierten Aufzählung `FileCompareMethod`. Über dieses Argument können Sie bestimmen, ob nur das Datum der letzten Änderung (`FileCompareMethod.Date`) oder nur der Inhalt (`FileCompareMethod.Content`) oder beides verglichen wird (`FileCompareMethod.DateAndContent`).

`CompareFiles` überprüft zunächst, ob die Dateinamen identisch sind, und beendet sich mit `true`, wenn dies der Fall ist. Danach vergleicht `CompareFiles` über zwei `FileInfo`-Objekte das Datum der letzten Änderung der Dateien, wenn am Argument `compareMethod` ein entsprechender Wert übergeben wurde. Ist dieses Datum unterschiedlich, kehrt `CompareFiles` mit der Rückgabe von `false` zurück. Das Datum der Erstellung der Datei, das Sie aus einem `FileInfo`-Objekt ebenfalls auslesen können, wird übrigens nicht verglichen. Der Grund dafür liegt darin, dass Windows (XP, 2000) beim Kopieren einer Datei von einem Ordner in einen anderen dieses Datum auf das aktuelle setzt, was den Vergleich der Erstelldaten untauglich macht.

Zum Vergleich des Inhalts öffnet `CompareFiles` die beiden Dateien über `FileStream`-Objekte und vergleicht direkt die Größe der Dateien. Ist diese unterschiedlich, sind die Dateien ungleich und die Methode wird mit der Rückgabe von `false` beendet. Damit ist sichergestellt, dass der dem Größenvergleich folgende zeitaufwändige byteweise Vergleich nur dann ausgeführt werden muss, wenn die Dateien dieselbe Größe besitzen.

Ist die Größe der Dateien identisch, liest `CompareFiles` die Streams in einer Schleife byteweise ein und vergleicht jedes Byte mit dem der anderen Datei. Die dazu verwendete Methode `ReadByte` gibt einen `int`-Wert zurück, der mit -1 aussagt, dass kein weiteres Byte mehr gelesen werden konnte. Im Schleifenfuß überprüft `CompareFiles` deswegen die Gleichheit der Werte und ob das aktuell gelesene Byte der ersten Datei -1 ist.

Nach dem Schließen der Streams wird dann nur noch überprüft, ob die beiden zuletzt ermittelten `int`-Werte gleich sind, und das Ergebnis dieses Vergleichs zurückgegeben.

`CompareFiles` erfordert die Einbindung der Namensräume `System` und `System.IO`.

```
/* Aufzählung für die möglichen Vergleichstypen */
public enum FileCompareMethod
{
   Date,
   Content,
   DateAndContent
}

/* Vergleich zwei Dateien */
public static bool CompareFiles(string filePath1, string filePath2,
   FileCompareMethod compareMethod)
{
   // Wenn beide Dateinamen identisch sind, true zurückgeben
   if (filePath1 == filePath2)
   {
      return true;
   }

   // Über FileInfo-Objekte das Datum der letzten Änderung vergleichen,
   // sofern dies gewünscht ist. Das Erstelldatum wird übrigens nicht
   // verglichen, weil dieses beim Kopieren von Dateien
   // auf das aktuelle Datum gesetzt wird
   if (compareMethod == FileCompareMethod.DateAndContent ||
      compareMethod == FileCompareMethod.Date)
   {
      FileInfo fi1 = new FileInfo(filePath1);
      FileInfo fi2 = new FileInfo(filePath2);
      if (fi1.LastWriteTime != fi2.LastWriteTime)
      {
         return false;
      }
   }

   // Den Inhalt vergleichen, sofern dies gewünscht ist
   if (compareMethod == FileCompareMethod.DateAndContent ||
      compareMethod == FileCompareMethod.Content)
   {
      // FileStream-Objekte für den Vergleich erzeugen
      FileStream fs1 = new FileStream(filePath1, FileMode.Open);
      FileStream fs2 = new FileStream(filePath2, FileMode.Open);
```

Listing 243: Methode zum Vergleich von zwei Dateien

```
      // Die Dateigröße vergleichen
      if (fs1.Length != fs2.Length)
      {
         return false;
      }

      // Die Dateien Byte für Byte vergleichen
      int fileByte1, fileByte2;
      do
      {
         fileByte1 = fs1.ReadByte();
         fileByte2 = fs2.ReadByte();
      } while (fileByte1 == fileByte2 && fileByte1 != -1);

      // Die Streams schließen
      fs1.Close();
      fs2.Close();

      // Das Ergebnis zurückgeben: Die Dateien sind gleich, wenn an dieser
      // Stelle die zuletzt gelesenen Bytes identisch sind
      return (fileByte1 == fileByte2);
   }

   return true;
}
```

Listing 243: Methode zum Vergleich von zwei Dateien (Forts.)

140 Ordner vergleichen

Das .NET Framework und das Windows-API bieten keine Möglichkeiten, zwei Ordner miteinander zu vergleichen. Ein solcher Vergleich muss wieder einmal selbst programmiert werden. Und das ist gar nicht so einfach.

Die Methode `CompareFolders` in Listing 244 implementiert den Vergleich zweier Ordner. Am ersten und zweiten Argument übergeben Sie die Ordnernamen, am dritten geben Sie an, ob die Methode den Inhalt der in den Ordnern enthaltenen Dateien vergleichen soll oder nur das Datum der letzten Änderung.

`CompareFolders` geht den am linken Argument angegebenen Ordner rekursiv durch und überprüft für den Ordner und jeden seiner Unterordner:

- ob die Anzahl der Unterordner im korrespondierenden anderen Ordner identisch ist,
- ob der korrespondierende andere Ordner dieselben Unterordner besitzt und
- ob der korrespondierende andere Ordner dieselben Dateien besitzt.

Dieser Methode werden je ein `DirectoryInfo`-Objekt für die aktuell zu vergleichenden Ordner und eine Info darüber übergeben, ob der Inhalt der Dateien im Ordner verglichen werden soll.

`CompareFolder` vergleicht zunächst über das `FileInfo`-Array, das die `GetFiles`-Methode der jeweiligen `DirectoryInfo`-Instanz zurückgibt, die Anzahl der Dateien in den beiden Ordnern und gibt `false` zurück, wenn diese unterschiedlich ist.

Danach geht `CompareFolder` alle Dateien des linken Ordners durch, ermittelt den korrespondierenden Dateinamen im rechten Ordner und überprüft, ob die Datei in diesem vorhanden ist. Wird eine nicht vorhandene Datei ermittelt, wird die Methode mit `false` beendet. Ist die Datei vorhanden, vergleicht `CompareFolder` je nach dem Argument `compareFileContent` entweder nur das Datum der letzten Änderung oder den Inhalt der Dateien. Zum Vergleich wird die Methode `CompareFiles` aus dem Rezept 139 eingesetzt. Das von dieser Methode erwartete Argument vom Typ `FileCompareMethod` wird dabei passend zum Wert des `CompareFolder`-Arguments eingestellt.

Die Anzahl der Unterordner und die Existenz der Unterordner des linken im rechten Ordner werden dann auf eine ähnliche Weise geprüft wie die der Dateien. Innerhalb der Schleife, die die Unterordner durchgeht, ruft `CompareFolder` sich nach der Überprüfung der Existenz des Unterordners dann für die jeweiligen Unterordner selbst auf um diese auf Gleichheit zu überprüfen. Dabei wird die Rückgabe des rekursiven Aufrufs ausgewertet und in dem Fall, dass die Methode `false` zurückgibt (weil die Unterordner nicht identisch sind), `return false` aufgerufen. Damit ist sichergestellt, dass der rekursive Aufruf möglichst schnell zurückkehrt. Am Ende gibt `CompareFolder` `true` zurück, weil an dieser Stelle (nach allen Überprüfungen) bewiesen ist, dass beide Ordner identisch sind.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
public static bool CompareFolder(DirectoryInfo folder1,
   DirectoryInfo folder2, bool compareFileContent)
{
   // Die Anzahl der Dateien im Ordner vergleichen
   FileInfo[] files1 = folder1.GetFiles();
   FileInfo[] files2 = folder2.GetFiles();
   if (files1.Length != files2.Length)
   {
      return false;
   }

   // Die Dateien durchgehen und diese vergleichen
   foreach (FileInfo file1 in files1)
   {
      // Dateiname im zweiten Ordner ermitteln und überprüfen, ob die Datei
      // existiert
      string fileName2 = Path.Combine(folder2.FullName, file1.Name);
      if (System.IO.File.Exists(fileName2))
      {
```

Listing 244: Methode zum Vergleichen zweier Ordner

```
            // Dateien vergleichen
            FileCompareMethod compareMethod =
               compareFileContent ? FileCompareMethod.Content :
               FileCompareMethod.Date;

            if (CompareFiles(file1.FullName, fileName2,
               compareMethod) == false)
            {
               return false;
            }
         }
         else

         {
            // Datei existiert nicht, also sind die Ordner nicht identisch
            return false;
         }
      }

      // Die Anzahl der Unterordner vergleichen
      DirectoryInfo[] subFolders1 = folder1.GetDirectories();
      DirectoryInfo[] subFolders2 = folder2.GetDirectories();
      if (subFolders1.Length != subFolders2.Length)
      {
         return false;
      }

      // Die Unterordner des ersten Ordners durchgehen um zu vergleichen, ob der
      // zweite Ordner dieselben Unterordner besitzt, und um diese rekursiv
      // durchzugehen
      foreach (DirectoryInfo subFolder1 in subFolders1)
      {
         // Ordnername des zweiten Ordners ermitteln und überprüfen, ob der
         // Ordner existiert
         string folderPath2 = Path.Combine(folder2.FullName, subFolder1.Name);
         DirectoryInfo subFolder2 = new DirectoryInfo(folderPath2);
         if (subFolder2.Exists == false)
         {
            return false;
         }

         // Rekursiver Aufruf zum Vergleich der beiden Unterordner
         if (CompareFolder(subFolder1, subFolder2,
            compareFileContent) == false)
         {
```

Listing 244: Methode zum Vergleichen zweier Ordner (Forts.)

```
            return false;
         }
      }

      // Wenn die Methode hier ankommt, enthalten beide (Unter-)Ordner einen
      // identischen Inhalt
      return true;
   }
```

Listing 244: Methode zum Vergleichen zweier Ordner (Forts.)

Zum vereinfachten Aufruf habe ich noch eine zweite Variante der `CompareFolders`-Methode entwickelt, der jeweils ein String übergeben wird, der den Pfad zum Ordner darstellt. Diese Variante ruft einfach die erste Variante mit zwei erzeugten `DirectoryInfo`-Objekten auf.

```
public static bool CompareFolders(string folderPath1, string folderPath2,
   bool compareFileContent)
{
   // Die übergebenen Ordnernamen angleichen
   if (folderPath1.EndsWith("\\") == false) folderPath1 += "\\";
   if (folderPath2.EndsWith("\\") == false) folderPath2 += "\\";

   // Die rekursive Vergleichsmethode mit zwei neuen DirectoryInfo-
   // Objekten für die beiden Ordner aufrufen
   DirectoryInfo folder1 = new DirectoryInfo(folderPath1);
   DirectoryInfo folder2 = new DirectoryInfo(folderPath2);
   return CompareFolder(folder1, folder2, compareFileContent);
}
```

Listing 245: Zweite Variante der Methode zum Vergleichen zweier Ordner

141 Die Größe einer Datei ermitteln

Zur Ermittlung der Größe einer Datei können Sie ein `FileInfo`-Objekt verwenden und dessen `Length`-Eigenschaft auslesen.

Das folgende Beispiel setzt ein solches Objekt ein und integriert eine Ausnahmebehandlung für den Fall, dass die Datei nicht existiert oder dass die Größen-Information nicht gelesen werden kann. Das Beispiel erfordert den Import der Namensräume `System` und `System.IO`.

```
string fileName = "c:\\pagefile.sys";

// Ermitteln der Größe der Datei
try
{
```

Listing 246: Ermitteln der Größe einer Datei

```
   // Größe über ein FileInfo-Objekt auslesen
   FileInfo fi = new FileInfo(fileName);
   long fileSize = fi.Length;

   Console.WriteLine("{0}: {1:#,#0} Byte.", fileName, fileSize);
}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Ermitteln der Dateigröße " +
      "der Datei '{0}': {1}.", fileName, ex.Message);
}
```

Listing 246: Ermitteln der Größe einer Datei (Forts.)

142 Dateiattribute auslesen

Wie Sie ja sicherlich wissen, können Dateien im Windows-Dateisystem mit Attributen ausgestattet sein. Das Attribut *ReadOnly* bestimmt zum Beispiel, dass die Datei schreibgeschützt ist.

Diese Attribute können Sie über die Eigenschaft `Attributes` eines `FileInfo`-Objekts (aus dem Namensraum `System.IO`) auslesen (und bearbeiten, wie ich in Rezept 143 zeige). `Attributes` ist ein Bitfeld, das aus den Konstanten der Aufzählung `FileAttributes` zusammengesetzt sein kann. Zur Überprüfung, ob bestimmte Attribute gesetzt sind, kombinieren Sie den Wert, der in `Attributes` gespeichert ist, über den bitweisen Und-Operator (&) mit den Attributen. Resultiert bei diesem Vergleich ein Wert größer Null, ist das Attribut gesetzt.

Listing 247, das den Import der Namensräume `System` und `System.IO` voraussetzt, zeigt diesen Vergleich am Beispiel der Datei *c:\pagefile.sys* für alle möglichen Dateiattribute (das Attribut `Directory`, das nur für Verzeichnisse gilt, wird hier nicht verwendet).

```
string fileName = "c:\\pagefile.sys";

// Ermitteln der Attribute der Datei
try
{
  // FileInfo-Objekt erzeugen
  FileInfo fi = new FileInfo(fileName);

  // Attribute auslesen
  Console.WriteLine("Dateiattribute von {0}", fileName);

  if ((fi.Attributes & FileAttributes.Archive) > 0)
  {
    Console.WriteLine("Archiv");
  }
```

Listing 247: Auslesen der Attribute einer Datei

```
if ((fi.Attributes & FileAttributes.Compressed) > 0)
{
  Console.WriteLine("Komprimiert");
}

if ((fi.Attributes & FileAttributes.Device) > 0)
{
   Console.WriteLine("Gerät (Dieses Attribut ist für " +
      "eine zukünftige Verwendung reserviert)");
}

if ((fi.Attributes & FileAttributes.Encrypted) > 0)
{
  Console.WriteLine("Verschlüsselt");
}

if ((fi.Attributes & FileAttributes.Hidden) > 0)
{
  Console.WriteLine("Versteckt");
}

if ((fi.Attributes & FileAttributes.Normal) > 0)
{
  Console.WriteLine("Normale Datei");
}

if ((fi.Attributes & FileAttributes. NotContentIndexed) > 0)
{
  Console.WriteLine("Nicht inhaltsindiziert");
}

if ((fi.Attributes & FileAttributes.Offline) > 0)
{
  Console.WriteLine("Offline");
}
if ((fi.Attributes & FileAttributes.ReadOnly) > 0)
{
  Console.WriteLine("Schreibgeschützt");
}

if ((fi.Attributes & FileAttributes.ReparsePoint) > 0)
{
  Console.WriteLine("Datei enthält einen Analysepunkt");
}
```

Listing 247: Auslesen der Attribute einer Datei (Forts.)

```
  if ((fi.Attributes & FileAttributes.SparseFile) > 0)
  {
    Console.WriteLine("Dünn besetzte Datei (mit vielen 0-Bytes)");
  }

  if ((fi.Attributes & FileAttributes.System) > 0)
  {
    Console.WriteLine("System");
  }

  if ((fi.Attributes & FileAttributes.Temporary) > 0)
  {
    Console.WriteLine("Temporäre Datei");
  }
}
catch (IOException ex)
{
  Console.WriteLine("Fehler beim Ermitteln der Dateiattribute " +
    "der Datei '{0}': {1}.", fileName, ex.Message);
}
```

Listing 247: Auslesen der Attribute einer Datei (Forts.)

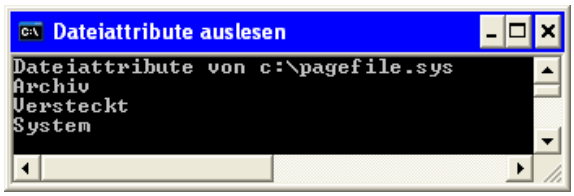


Abbildung 80: Ermittlung der Dateiattribute der Datei c:\pagefile.sys

143 Dateiattribute (z. B. den Schreibschutz) setzen oder aufheben

Windows verwendet verschiedene Attribute zur Verwaltung von Meta-Informationen zu einer Datei. Das Attribut *ReadOnly* zum Beispiel bestimmt, ob eine Datei schreibgeschützt ist.

Die Attribute einer Datei können Sie über die `Attributes`-Eigenschaft eines `FileInfo`-Objekts sehr einfach setzen oder aufheben. Diese Eigenschaft ist ein Bitfeld, das aus den Konstanten der Aufzählung `FileAttributes` besteht. Über ein wenig Bit-Arithmetik können Sie die Attribute setzen oder entfernen.

Zum Setzen eines Attributs kombinieren Sie den Wert, den die `Attributes`-Eigenschaft verwaltet, über ein bitweises Oder (|) mit den zu setzenden Attributen. Zum Entfernen verwenden Sie das bitweise Und (&) mit den negierten Attributen. Listing 248 zeigt, wie Sie dies programmieren können.

```
/* Setzt Dateiattribute für eine Datei */
public static void SetFileAttributes(string filename,
   FileAttributes attributesToSet)
{
   FileInfo fileInfo = new FileInfo(filename);
   fileInfo.Attributes = fileInfo.Attributes | attributesToSet;
}

/* Entfernt Dateiattribute von einer Datei */
public static void RemoveFileAttributes(string filename,
   FileAttributes attributesToRemove)
{
   FileInfo fileInfo = new FileInfo(filename);
   fileInfo.Attributes = fileInfo.Attributes & ~attributesToRemove;
}
```

Listing 248: Methoden zum Setzen und zum Entfernen von Attributen einer Datei

144 Die Erstell- und Zugriffsdaten einer Datei lesen und setzen

Windows verwaltet für alle Dateien das Datum, an dem die Datei erstellt wurde, das Datum, an dem der letzte Zugriff erfolgte, und das Datum, an dem die Datei zuletzt geändert wurde. Diese Datumswerte können Sie über ein `FileInfo`-Objekt lesen und auch ändern. Dazu verwenden Sie die Eigenschaften `CreationTime` (Erstelldatum), `LastAccessTime` (Datum des letzten Zugriffs) und `LastWriteTime` (Datum der letzten Änderung).

Das folgende Konsolenanwendungs-Beispiel liest das Erstelldatum und die Zugriffsdaten der Datei *Demo.txt* im Programmverzeichnis und ändert das Erstelldatum auf den 1.1.2006 00:00:00. Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Reflection` und `System.IO` importieren.

```
string applicationPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string filename = Path.Combine(applicationPath, "Demo.txt");

// Erstell- und Zugriffsdaten ermitteln
FileInfo fi = new FileInfo(filename);
Console.WriteLine("Erstelldatum: " + fi.CreationTime);
Console.WriteLine("Letzter Zugriff: " + fi.LastAccessTime);
Console.WriteLine("Letzte Änderung: " + fi.LastWriteTime);

// Erstelldatum ändern
fi.CreationTime = new DateTime(2006, 1, 1);
```

Listing 249: Auslesen der Erstell- und Zugangsdaten einer Datei und Ändern des Erstelldatums

145 Die Version einer Datei auslesen

Unter Windows werden alle Dateien mit Versionsinformationen versehen, die Sie z. B. im Explorer in den Eigenschaften einer Datei auslesen können. Die Version besteht aus vier 16 Bit großen Zahlen, die intern zu einem 64-Bit-Wert verknüpft werden.

Die erste 16-Bit-Zahl bestimmt die Haupt- (Major) und die zweite die Nebenversion (Minor). Haupt- und Nebenversion identifiziert nach der Microsoft-Konvention (wenigstens für Assemblys, siehe in der Dokumentation der `Version`-Klasse) große Änderungen in einer Datei (hauptsächlich in einem Programm oder einer Klassenbibliothek o. Ä.). Änderungen in der Hauptversion stehen dafür, dass eine Datei nicht mehr zu älteren Versionen kompatibel ist, Änderungen in der Nebenversion deuten auf eine eventuell noch vorhandene Rückwärts-Kompatibilität hin.

Die dritte Zahl steht für die Build-Nummer. Änderungen in dieser Nummer stehen für eine Neukompilierung desselben Programmcodes, lediglich mit geänderten Compilereinstellungen. Eine Erhöhung der Build-Nummer sollte also keine Inkompatibilitäten bewirken.

Die letzte Zahl verwaltet die Revisions-Nummer. Dateien mit unterschiedlichen Revisionsnummern sollten vollständig austauschbar sein. Diese Nummer ist hauptsächlich für Bugfixes und für die Beseitigung von Sicherheitsmängeln vorgesehen.

Um die Versionsinformationen einer Datei zu ermitteln, können Sie eine Instanz der Klasse `FileVersionInfo` aus dem Namensraum `System.Diagnostics` verwenden. Die statische Methode `GetVersionInfo` dieser Klasse gibt unter der Übergabe eines Dateinamens eine `FileVersionInfo`-Instanz zurück. Über verschiedene Eigenschaften erhalten Sie Informationen zur Version, wie es Listing 250 zeigt. Das Programm erfordert den Import der Namensräume `System`, `System.IO` und `System.Diagnostics`.

```
// Vollen Dateinamen der Datei kernel32.dll ermitteln
string fileName = Path.Combine(Environment.SystemDirectory,
   "kernel32.dll");

// Versionsinformationen auslesen
FileVersionInfo fileVersionInfo = FileVersionInfo.GetVersionInfo(fileName);

// Version ausgeben
Console.WriteLine("Version der Datei {0}", fileName);
Console.WriteLine("Major: {0}", fileVersionInfo.FileMajorPart);
Console.WriteLine("Minor: {0}", fileVersionInfo.FileMinorPart);
Console.WriteLine("Build: {0}", fileVersionInfo.FileBuildPart);
Console.WriteLine("Revision: {0}", fileVersionInfo.FilePrivatePart);
Console.WriteLine("Version: {0}", fileVersionInfo.FileVersion);
```

Listing 250: Auslesen der Versionsinformationen einer Datei

Die Ausgabe des Programms zeigt Abbildung 81.

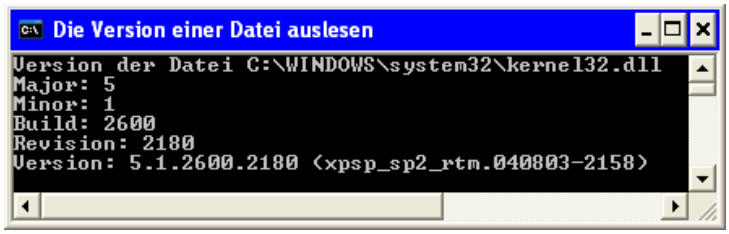


Abbildung 81: Das Beispielprogramm zum Auslesen von Versionsinformationen in Aktion

146 Den kurzen (8.3-)Dateinamen einer Datei auslesen

Windows verwaltet für jede Datei neben dem normalen (langen) Dateinamen einen kurzen, zu alten DOS-Programmen kompatiblen Dateinamen. Dieser besteht aus einem acht Zeichen großen Namen und einer dreistelligen Endung. Der Name wird am Ende automatisch mit einer Tilde und einer Ziffer versehen, um eine Eindeutigkeit bei langen Dateinamen sicherzustellen, die mit denselben Zeichen beginnen.

Wenn Sie den kurzen Dateinamen einer Datei (inklusive Pfad) ermitteln wollen, weil Sie z. B. mit einem alten DOS-Programm kommunizieren müssen, finden Sie im .NET Framework hierfür keine Möglichkeit. Dazu müssen Sie die Windows-API-Funktion `GetShortPathName` verwenden, der Sie den langen Dateinamen inklusive Pfad übergeben. `GetShortPathName` wird folgendermaßen deklariert:

```
[DllImport("kernel32.dll", SetLastError=true)]
private static extern int GetShortPathName(string lpszLongPath,
   System.Text.StringBuilder lpszShortPath, int cchBuffer);
```

Am Argument `lpszLongPath` übergeben Sie den langen Dateinamen. Das Argument `lpszShortPath` muss mit einem für Pfadangaben ausreichend groß dimensionierten `StringBuilder`-Objekt belegt werden. Im letzten Argument übergeben Sie die Größe dieses Objekts. `GetShortPathName` ermittelt an diesem Argument, wie viele Zeichen in den String geschrieben werden können.

Die Methode `GetShortName` in Listing 251 setzt diese Funktion zur Ermittlung des kurzen Dateinamens einer Datei ein. `GetShortName` ruft `GetShortPathName` mit dem übergebenen Dateinamen und einem `StringBuilder`-Objekt auf, das auf 261 Zeichen Größe initialisiert wurde (die maximale Pfadgröße plus Platz für das 0-Zeichen, mit dem die in C oder C++ geschriebene Funktion den String abschließt). Dazu ist anzumerken, dass `GetShortPathName` nicht nur den Dateinamen selbst, sondern auch den kompletten Pfad zurückgibt, wenn auch der Quelldateiname einen Pfad beinhaltet.

Im Erfolgsfall gibt `GetShortPathName` die Anzahl der gelesenen Zeichen zurück. In diesem Fall gibt `GetShortName` den ermittelten Dateinamen direkt zurück. Im Fehlerfall, bei dem `GetShortPathName` 0 zurückgibt, liest `GetShortName` den letzten Windows-Fehlercode aus und ermittelt mit diesem Code über `FormatMessage` eine Fehlerbeschreibung. Damit der Windows-Fehlercode überhaupt verfügbar ist, habe ich `GetShortPathName` mit `SetLastError=true` im `DllImport`-Attribut deklariert. Mit der so ermittelten (aussagekräftigen) Fehlerbeschreibung erzeugt `GetShortName` dann eine Ausnahme vom Typ `IOException`.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration der benötigten API-Funktionen und -Konstanten */
[DllImport("kernel32.dll", SetLastError=true)]
private static extern int GetShortPathName(string lpszLongPath,
   StringBuilder lpszShortPath, int cchBuffer);

[DllImport("Kernel32.dll")]
private static extern int FormatMessage(int dwFlags, string lpSource,
   int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
   string [] Arguments);

private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;
private const int MAX_PATH = 260;

/* Methode zur Ermittlung des kurzen Dateinamens einer Datei */
public static string GetShortName(string fileName)
{
   // GetShortPathName mit einem ausreichend großen StringBuilder-Objekt
   // aufrufen
   StringBuilder shortName = new StringBuilder(MAX_PATH + 1);
   if (GetShortPathName(fileName, shortName, MAX_PATH + 1) > 0)
   {
      return shortName.ToString();
   }
   else
   {
      // Fehler bei der Ausführung: API-Fehler ermitteln, diesen in eine
      // Fehlerbeschreibung umwandeln und damit eine Ausnahme werfen
      System.Text.StringBuilder errorMessage = new StringBuilder(1024);
      FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, null,
         Marshal.GetLastWin32Error(), 0, errorMessage, 1024, null);
      throw new IOException(errorMessage.ToString());
   }
}
```

Listing 251: Methode zum Auslesen des kurzen Dateinamens einer Datei

Die Anwendung dieser Methode ist sehr einfach. Sie sollten natürlich eine Ausnahmebehandlung vorsehen, für den Fall, dass die Datei nicht existiert oder ein sonstiger Fehler auftritt. Abbildung 82 zeigt ein Beispielprogramm, das über `GetShortName` den kurzen Namen der Datei *readme.htm* im Ordner von Visual Studio ermittelt hat.



Abbildung 82: Ermittlung des kurzen Dateinamens der Visual-Studio-Readme-Datei

147 Alle Dateien eines Ordners auflisten

Um alle Dateien eines Ordners aufzulisten, können Sie ein `DirectoryInfo`-Objekt (aus dem Namensraum `System.IO`) verwenden und über dessen `GetFiles`-Methode Informationen über alle enthaltenen Dateien auslesen. `GetFiles` gibt ein Array mit `FileInfo`-Objekten zurück, das Sie sehr einfach durchgehen können. Aus der Eigenschaft `Name` der `FileInfo`-Objekte können Sie den Dateinamen auslesen. Für den Fall, dass ein ungültiger Ordner angegeben ist oder ein anderer Fehler auftritt, sollten Sie eine Ausnahmebehandlung vorsehen:

```
// Dateiname festlegen
string folderName = "c:\\";

try
{
   // DirectoryInfo-Objekt erzeugen
   DirectoryInfo di = new DirectoryInfo(folderName);

   // Dateien ermitteln
   FileInfo[] files = di.GetFiles();

   // Alle Dateien durchgehen und den Namen ausgeben
   foreach (FileInfo file in files)
   {
      Console.WriteLine(file.Name);
   }
}
catch (IOException ex)
{
   Console.WriteLine("Fehler beim Ermitteln der Dateien: {0}.", ex.Message);
}
```

Listing 252: Ermitteln aller Dateien eines Ordners

148 Dateien und Ordner mit Fortschrittsdialog, Abbruchmöglichkeit und Überschreib-Nachfrage umbenennen oder verschieben

Das Umbenennen und Verschieben einer Datei bzw. eines Ordners über die `Move`-Methode der `File`- bzw. der `Directory`-Klasse ermöglicht leider kein Überschreiben vorhandener Dateien. Wenn Sie dies dem Anwender ermöglichen wollen, müssen Sie die Windows-API-Funktion `SHFileOperation` verwenden, die ich bereits im Rezept 125 für das Kopieren von Ordnern beschrieben habe. Damit können Sie, wie bereits beim Kopieren, dem Anwender die Entscheidung überlassen, bereits vorhandene Dateien bzw. Ordner zu überschreiben (Abbildung 83), und zudem einen automatischen Fortschrittsdialog anzeigen lassen.



Abbildung 83: Der Überschreib-Bestätigungsdialog von SHFileOperation beim Verschieben eines Ordners

Die Methode `MoveFileOrFolder` in Listing 253 zum Verschieben einer Datei bzw. eines Ordners entspricht fast komplett der Methode `CopyFolder` zum Kopieren von Ordnern aus dem Rezept Listing 53. Als einziger Unterschied wird als Funktion für `SHFileOperation` nun `FO_MOVE` im Feld `wFunc` angegeben.

Im Argument `source` geben Sie den Pfad zur Quelldatei bzw. den Pfad zum Quellordner an. Das Argument `dest` bestimmt den Zielordner bzw. die Zieldatei. `confirmOverwrites` bestimmt, ob der Anwender vor dem Überschreiben von bereits vorhandenen Dateien und Ordnern vom System gefragt wird, ob er diese wirklich überschreiben will.

`MoveFileOrFolder` hat mit demselben Problem zu kämpfen wie `CopyFolder`, nämlich dass die Rückgabe von `SHFileOperation` nicht dokumentiert ist. Deswegen kann diese Methode leider nur auswerten, ob das Verschieben korrekt ausgeführt wurde. Gibt `SHFileOperation` einen Wert ungleich Null zurück, ist das Verschieben fehlgeschlagen. Da es dabei auch vorkommen kann, dass der Benutzer den Vorgang abgebrochen hat, wirft `MoveFileOrFolder` keine Exception, sondern gibt lediglich einen booleschen Wert zurück.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration benötigter API-Funktionen, -Konstanten und -Strukturen */
[DllImport("shell32.dll", CharSet = CharSet.Auto)]
private static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

private const int FO_MOVE = 0x0001;            // Verschieben
private const int FOF_ALLOWUNDO = 0x0040;      // Undo ermöglichen
private const int FOF_NOCONFIRMATION = 0x0010; // Keine Nachfrage
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
private struct SHFILEOPSTRUCT
{
   public IntPtr hwnd;
   public int wFunc;
   public string pFrom;
   public string pTo;
   public short fFlags;
   [MarshalAs(UnmanagedType.Bool)]
   public bool fAnyOperationsAborted;
   public IntPtr hNameMappings;
   public string lpszProgressTitle;
}

/* Verschiebt eine Datei */
public static bool MoveFileOrFolder(string source, string dest,
   bool confirmOverwrites)
{
   // Struktur für die Dateiinformationen erzeugen
   SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

   // (Unter-)Funktion definieren (ShFileOperation kann auch Dateien und
   // Ordner löschen und kopieren)
   fileOp.wFunc = FO_MOVE;

   // Quelle und Ziel definieren. Dabei müssen mehrere Datei- oder
   // Ordnerangaben über 0-Zeichen getrennt werden.
   // Am Ende muss ein zusätzliches 0-Zeichen stehen
   fileOp.pFrom = source + "\x0\x0";
   fileOp.pTo = dest + "\x0\x0";

   // Flags setzen, sodass ein Rückgängigmachen möglich ist (was aber
   // beim Verschieben zurzeit scheinbar noch nicht unterstützt wird!)
   // und - je nach Argument confirmOverwrites - keine Nachfrage
   // beim Überschreiben von Ordnern beim Anwender erfolgt
   if (confirmOverwrites)
   {
      fileOp.fFlags = FOF_ALLOWUNDO;
```

Listing 253: Methode zum Verschieben einer Datei oder eines Ordners über SHFileOperation

```
    }
    else
    {
        fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;
    }

    // SHFileOperation unter Übergabe der Struktur aufrufen
    int result = SHFileOperation(ref fileOp);

    // Erfolg oder Misserfolg zurückgeben
    return (result == 0);
}
```

Listing 253: Methode zum Verschieben einer Datei oder eines Ordners über SHFileOperation (Forts.)

Hinweis

Beim Verschieben (und Kopieren) von Ordnern verhält sich `SHFileOperation` etwas gewöhnungsbedürftig: Existiert noch kein Ordner mit dem angegebenen Zielpfad, wird der Quellordner in den im Zielpfad angegebenen übergeordneten Ordner verschoben und so benannt, wie der letzte Ordner im Zielpfad. Das ist das erwartete Verhalten. Existiert jedoch bereits ein Ordner mit dem angegebenen Zielpfad, wird der Quellordner so in diesem Ordner verschoben, dass er ein Unterordner wird. Existiert z. B. ein Ordner *C:\Temp\DemoFolder* und Sie verschieben den Ordner *C:\DemoFolder* nach *C:\Temp\DemoFolder*, wird dieser Ordner in den Ordner *C:\Temp\DemoFolder\DemoFolder* verschoben. Existiert *C:\Temp\DemoFolder* beim Verschieben noch nicht, wird der Ordner korrekt in den Ordner *C:\Temp\DemoFolder* verschoben. Deshalb sollten Sie beachten, dass Sie beim Verschieben von Ordnern als Ziel immer den Pfad zu dem Ordner angeben sollten, in den der verschobene Ordner kopiert werden soll (also den Zielpfad zum übergeordneten Ordner). Leider führt das dazu, dass Sie Ordner beim Verschieben nicht umbenennen können.

Bei der Anwendung dieser Methode können Sie entweder Datei- oder Ordnerpfade übergeben. Das folgende Beispiel verschiebt den Ordner *c:\demofolder* (der für das Beispiel gegebenenfalls automatisch angelegt wird) nach *c:\temp\demofolder* und die Datei *c:\Demo for API-Move.txt* (die ebenfalls gegebenenfalls automatisch angelegt wird) nach *c:\temp\Demo for API-Move.txt*.

```
// Ordner verschieben
string sourceFolderName = @"c:\DemoFolder";    // Quellordner
string destFolderName = @"c:\Temp"; // Zielordner (ohne Ordnername)

// Ordner als Beispiel zum Verschieben anlegen
System.IO.Directory.CreateDirectory(sourceFolderName);
```

Listing 254: Anwendung der Methode MoveFileOrFolder zum komfortablen Verschieben von Dateien oder Ordnern

```
// Ordner rekursiv mit Anwender-Nachfrage für das Überschreiben von
// Ordnern verschieben
Console.WriteLine("Verschiebe {0} nach {1} ... ", sourceFolderName,
   destFolderName);
if (MoveFileOrFolder(sourceFolderName, destFolderName, true))
{
   Console.WriteLine("Ordner erfolgreich verschoben");
}
else
{
   Console.WriteLine("Fehler beim Verschieben des Ordners");
}

// Datei verschieben
string sourceFileName = @"c:\Demo for API-Move.txt";    // Quelldatei
string destFileName = @"c:\Temp\Demo for API-Move.txt"; // Zieldatei

// (Text-)Datei als Beispiel zum Verschieben erzeugen
System.IO.StreamWriter sw = System.IO.File.CreateText(sourceFileName);
sw.WriteLine("Beispieldatei für das Verschieben einer Datei " +
   "über SHFileOperation.");
sw.Close();

// Datei mit Anwender-Nachfrage für das Überschreiben von Dateien verschieben
Console.WriteLine("Verschiebe {0} nach {1} ... ", _
   sourceFileName, destFileName);
if (MoveFileOrFolder(sourceFileName, destFileName, true))
{
   Console.WriteLine("Datei erfolgreich verschoben");
}
else
{
   Console.WriteLine("Fehler beim Verschieben der Datei");
}
```

Listing 254: Anwendung der Methode MoveFileOrFolder zum komfortablen Verschieben von Dateien oder Ordnern (Forts.)

Abbildung 84 und Abbildung 85 zeigen dieses Programm für den Fall, dass der Zielordner und die Zieldatei bereits existieren.

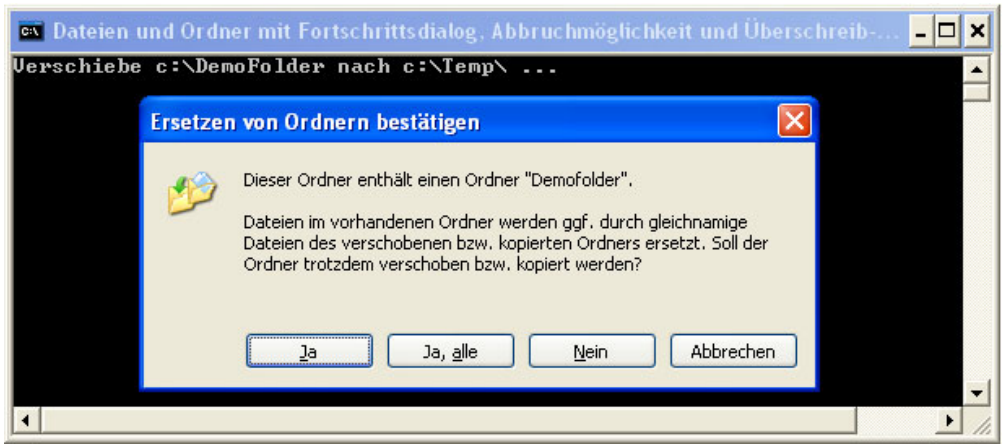


Abbildung 84: Verschieben eines Ordners, der im Ziel bereits existiert

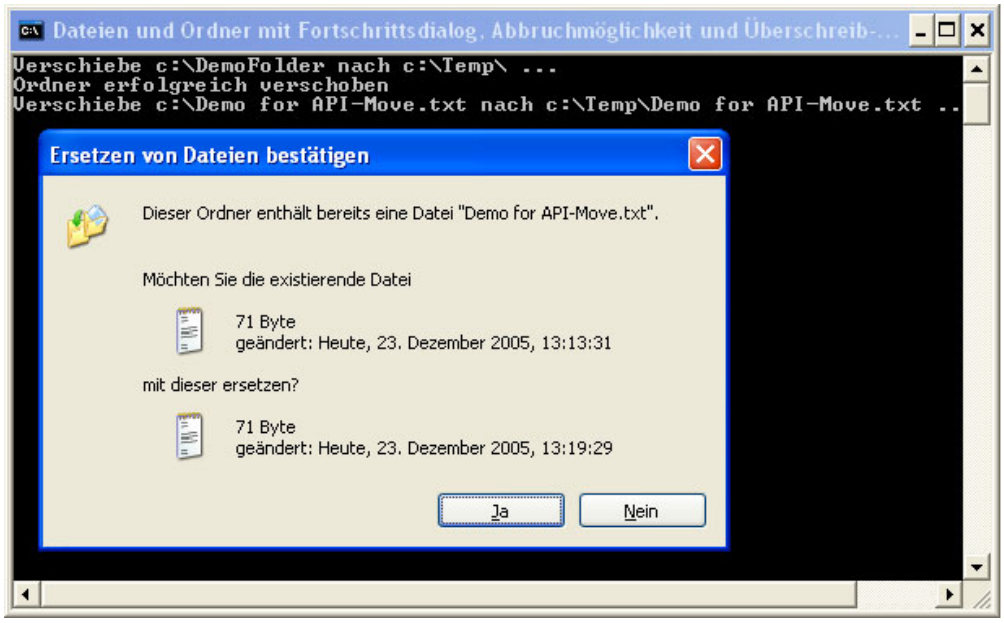


Abbildung 85: Verschieben einer Datei, die im Ziel bereits existiert

149 Dateien und Ordner in den Papierkorb verschieben

Das .NET Framework unterstützt das Verschieben von Dateien und Ordnern in den Papierkorb leider nicht direkt. Dazu müssen Sie die Windows-API-Funktion `SHFileOperation` einsetzen, die Dateien und Ordner kopieren, verschieben, umbenennen und (u. a. in den Papierkorb) löschen kann. Diese Funktion, die ich bereits in den Rezepten 125 und 148 eingesetzt habe, steht ab Windows 95 zur Verfügung.

SHFileOperation erwartet als Argument eine als Referenz zu übergebende, normale (C-)Struktur mit Informationen zu der auszuführenden Aktion. In dieser Struktur geben Sie die auszuführende Aktion und die Datei- bzw. Ordnernamen an. In einem »Flags«-Feld können Sie die Aktion näher spezifizieren. Für das Löschen einer Datei bzw. eines Ordners können Sie hier z. B. festlegen, dass der Vorgang rückgängig gemacht werden kann (womit die Datei bzw. der Ordner automatisch in den Papierkorb verschoben wird) und dass Windows beim Anwender nicht nachfragt, ob die Datei oder der Ordner wirklich verschoben werden soll.

Die Methode MoveToRecycleBin in Listing 255 implementiert das Verschieben in den Papierkorb über SHFileOperation. Als Argument erwartet diese Funktion lediglich den Pfad zu der Datei bzw. zum Ordner, die bzw. der gelöscht werden soll.

Das Prinzip dieser Methode entspricht dem, das ich bereits im Rezept Listing 55 erläutert habe. Die wesentlichen Unterschiede zu diesem Rezept sind die folgenden:

Zunächst schreibt MoveToRecycleBin zur Definition der Unterfunktion von SHFileOperation die Konstante FO_DELETE in das Feld wFunc der SHFILEOPSTRUCT-Struktur (die an SHFileOperation übergeben wird), um zu definieren, dass die Datei bzw. der Ordner gelöscht werden soll. Das Feld fFlags wird mit der Konstante FOF_ALLOWUNDO beschrieben. Damit werden gelöschte Dateien bzw. Ordner automatisch in den Papierkorb verschoben. Da das Ziel in diesem Fall fest definiert ist, wird nur das Feld pFrom mit dem übergebenen Pfad definiert.

Aufgrund der Problematik bei der Auswertung von Fehlern bei der Verwendung von SHFileOperation (siehe Rezept 125) habe ich darauf verzichtet, im Fehlerfall eine Ausnahme zu erzeugen. Da das Flag FOF_NOERRORUI nicht gesetzt wird, werden Fehler von SHFileOperation automatisch in einem Dialog gemeldet. MoveToRecycleBin meldet über einen booleschen Wert lediglich zurück, ob das Löschen in den Papierkorb erfolgreich war oder nicht.

Zum Kompilieren dieser Methode müssen Sie die Namensräume System und System.Runtime.InteropServices importieren.

```
/* Deklaration benötigter API-Funktionen, -Konstanten und -Strukturen */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

private const int FO_DELETE = 0x0003;          // Datei/Ordner löschen
private const int FOF_ALLOWUNDO  = 0x0040;     // Undo ermöglichen
private const int FOF_NOCONFIRMATION = 0x0010; // keine Nachfrage

[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
public struct SHFILEOPSTRUCT
{
   public IntPtr hwnd;
   public int wFunc;
   public string pFrom;
   public string pTo;
   public short fFlags;
```

Listing 255: Methode zum Verschieben von Dateien und Ordnern in den Papierkorb

```
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
}

/* Methode zum Verschieben einer Datei in den Papierkorb */
public static bool MoveToRecycleBin(string path)
{
    // Struktur für die Dateiinformationen erzeugen
    SHFILEOPSTRUCT fileOp = new SHFILEOPSTRUCT();

    // Quelle definieren. Dabei müssen mehrere Datei- oder
    // Ordnerangaben über 0-Zeichen getrennt werden.
    // Am Ende muss ein zusätzliches 0-Zeichen stehen
    fileOp.pFrom = path + "\x0\x0";

    // Flags setzen, sodass ein Rückgängigmachen möglich ist und
    // keine Nachfrage beim Anwender erfolgt
    fileOp.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;

    // (Unter-)Funktion definieren
    fileOp.wFunc = FO_DELETE;

    // ShFileOperation unter Übergabe der Struktur aufrufen
    int result = SHFileOperation(ref fileOp);

    // Erfolg oder Fehler zurückmelden. SHFileOperation liefert 0
    // zurück, wenn kein Fehler aufgetreten ist
    return (result == 0);
}
```

Listing 255: Methode zum Verschieben von Dateien und Ordnern in den Papierkorb (Forts.)

Listing 256 zeigt eine Anwendung, die über `MoveToRecycleBin` die Datei *c:\autoexec.bat.copy* und den Ordner *c:\temp\RecycleBinDemoFolder* in den Papierkorb verschiebt.

```
// Datei als Beispiel erst kopieren
File.Copy(@"c:\autoexec.bat", @"c:\autoexec.bat.copy", true);

// Ordner als Beispiel erst erzeugen
Directory.CreateDirectory(@"C:\Temp\RecycleBinDemoFolder");

// Datei in den Papierkorb verschieben
if (MoveToRecycleBin(@"c:\autoexec.bat.copy"))
```

Listing 256: Verschieben einer Datei und eines Ordners in den Papierkorb

```
{
   Console.WriteLine("Datei erfolgreich in den Papierkorb verschoben.");
}
else
{
   Console.WriteLine("Datei nicht erfolgreich in den Papierkorb " +
      "verschoben.");
}

// Ordner in den Papierkorb verschieben
if (MoveToRecycleBin(@"C:\Temp\RecycleBinDemoFolder"))
{
   Console.WriteLine("Ordner erfolgreich in den Papierkorb verschoben.");
}
else
{
   Console.WriteLine("Ordner nicht erfolgreich in den Papierkorb " +
      "verschoben.");
}
```

Listing 256: Verschieben einer Datei und eines Ordners in den Papierkorb (Forts.)

150 Größe des Papierkorbs und Anzahl der Dateien im Papierkorb ermitteln

Über die Windows-API-Funktion `SHQueryRecycleBin` können Sie ab Windows 95 die Größe des Papierkorbs und die Anzahl der enthaltenen Dateien eines Laufwerks oder aller Laufwerke ermitteln. `SHQueryRecycleBin` erwartet am ersten Argument den Wurzel-Pfad des Laufwerks, dessen Papierkorb erforscht werden soll, oder einen leeren String, wenn alle Papierkörbe berücksichtigt werden sollen (was, wie ich gleich noch beschreibe, unter Windows 2000 leider nicht funktioniert). Am zweiten Argument übergeben Sie eine Struktur vom Typ `SHQUERYRBINFO`, in der die Funktion die gelesenen Werte zurückgibt.

Die Methode `GetRecycleBinInfo` in Listing 57 implementiert das Lesen der Papierkorb-Daten. `GetRecycleBinInfo` erwartet einen String, der entweder leer oder `null` ist oder den Stammordner des Laufwerks bestimmt, dessen Papierkorb ausgelesen werden soll. Die Methode gibt eine Instanz der Klasse `RecycleBinInfo` mit den gelesenen Informationen zurück.

Unter Windows 2000 führt der Aufruf von `SHQueryRecycleBin` leider zu dem Fehler `E_INVALIDARG` (0x80070057), wenn `null` oder ein leerer String übergeben wird. Dieser Bug ist im Microsoft Knowledge Base-Artikel 266366 dokumentiert.

Um dieses Problem zu umschiffen, geht `GetRecycleBinInfo` bei der Übergabe eines Leerstrings oder von `null` alle Festplatten-Laufwerke des Systems durch und speichert deren Wurzelpfade in der `List`-Instanz `drives`. Die hierbei verwendete Technik basiert auf dem Rezept 120. Wird ein nicht leerer String übergeben, speichert `GetRecycleBinInfo` lediglich diesen in `drives`.

Danach wird die `RecycleBinInfo`-Instanz für die Rückgabe und die Instanz der `SHQUERYRBINFO`-Struktur erzeugt. Da `SHQueryRecycleBin` die Größe der Struktur im Feld `cbSize` erwartet (um

diese beim Schreiben zu kennen), liest GetRecycleBinInfo die Größe über Marshal.SizeOf so aus, dass die tatsächlich an die externe Funktion übergebene Größe ermittelt wird.

Danach geht GetRecycleBinInfo alle ermittelten Laufwerke durch, liest über SHQueryRecycleBin die Informationen der einzelnen Papierkörbe und addiert die gelesenen Werte zu den bereits in der RecycleBinInfo-Instanz gespeicherten. Wenn nur die Daten des Papierkorbs eines Laufwerks ausgelesen werden sollen, beschränkt sich die Schleife natürlich auf einen Durchlauf. Auf diese Weise wird das Windows-2000-Problem aber mehr oder weniger elegant umgangen ☺.

GetRecycleBinInfo gibt die RecycleBinInfo-Instanz schließlich zurück. Zum Kompilieren dieser Methode müssen Sie die Namensräume System, System.Collections.Generic und System.Runtime.InteropServices importieren.

```
/* Klasse für die Rückgabe der GetRecycleBinInfo-Methode */
public class RecycleBinInfo
{
   public long Size = 0;
   public long Count = 0;
}

/* Struktur, in der SHQueryRecycleBin die gelesenen Daten zurückgibt */
[StructLayout(LayoutKind.Sequential, Pack=1)]
   private struct SHQUERYRBINFO
{
   public int cbSize;
   public long i64Size;
   public long i64NumItems;
}

/* Deklaration der benötigten API-Funktionen und -Konstanten */
[DllImport("shell32.dll", CharSet=CharSet.Auto)]
static extern int SHQueryRecycleBin(string rootPath,
   ref SHQUERYRBINFO shQueryRBInfo);

[DllImport("Kernel32.dll")]
private static extern uint GetDriveType(string lpRootPathName);

private const uint DRIVE_FIXED = 3;

/* Methode zum Auslesen der Anzahl und der Größe der Dateien
   im Papierkorb */
public static RecycleBinInfo GetRecycleBinInfo(string rootPath)
{
   // Die auszulesenden Laufwerke bestimmen
   List<string> drives = new List<string>();
```

Listing 257: Klasse mit einer Methode zum Ermitteln der Anzahl der Elemente im Papierkorb und der Gesamtgröße der enthaltenen Dateien

```
if (rootPath != null && rootPath != "")
{
   drives.Add(rootPath);
}
else
{
   // Alle Festplatten-Partitionen ermitteln
   string[] logicalDrives = Environment.GetLogicalDrives();
   for (int i = 0; i < logicalDrives.Length; i++)
   {
      if (GetDriveType(logicalDrives[i]) == DRIVE_FIXED)
      {
         drives.Add(logicalDrives[i]);
      }
   }
}

// RecycleBinInfo-Instanz für das Ergebnis und SHQUERYRBINFO-Struktur
// erzeugen
RecycleBinInfo rbi = new RecycleBinInfo();
SHQUERYRBINFO shQueryRBInfo = new SHQUERYRBINFO();

// Größe der Struktur definieren, damit die API-Funktion diese beim
// Schreiben kennt
shQueryRBInfo.cbSize = Marshal.SizeOf(typeof(SHQUERYRBINFO));

// Alle ermittelten Laufwerke durchgehen
for (int i = 0; i < drives.Count; i++)
{
   // SHQueryRecycleBin aufrufen und eventuelle Fehler auswerten
   int result = SHQueryRecycleBin(drives[i] + "\x0",
      ref shQueryRBInfo);
   if (result == 0)
   {
      // Kein Fehler: Die gelesenen Daten mit dem Ergebnis addieren
      rbi.Size += shQueryRBInfo.i64Size;
      rbi.Count += shQueryRBInfo.i64NumItems;
   }
   else
   {
      // Fehler beim Lesen: Ausnahme erzeugen
      throw new Exception(String.Format(
         "Fehler 0x{0:X} beim Lesen des Papierkorbs", result));
   }
```

Listing 257: Klasse mit einer Methode zum Ermitteln der Anzahl der Elemente im Papierkorb und der Gesamtgröße der enthaltenen Dateien (Forts.)

```
    }

    return rbi;
}
```

Listing 257: Klasse mit einer Methode zum Ermitteln der Anzahl der Elemente im Papierkorb und der Gesamtgröße der enthaltenen Dateien (Forts.)

Listing 258 zeigt den maßgeblichen Quelltext einer Konsolenanwendung, die über `GetRecycleBinInfo` die Größe des gesamten Papierkorbs und die des Papierkorbs des Laufwerks C: ermittelt.

```
// Anzahl und Größe der Dateien und Größe des gesamten Papierkorbs ermitteln
RecycleBinInfo rbi = GetRecycleBinInfo(null);
Console.WriteLine("Anzahl der Dateien im gesamten Papierkorb: {0}",
   rbi.Count);
Console.WriteLine("Größe des gesamten Papierkorbs: {0} Byte", rbi.Size);
Console.WriteLine();

// Anzahl der Dateien und Größe des Papierkorbs auf C: ermitteln
rbi = GetRecycleBinInfo("C:\\");
Console.WriteLine("Anzahl der Dateien im Papierkorb auf C: {0}",
   rbi.Count);
Console.WriteLine("Größe des Papierkorbs auf C: {0} Byte", rbi.Si
```

Listing 258: Beispielhafte Anwendung der Methode GetRecycleBinInfo

In Abbildung 86 sehen Sie diese Anwendung in Aktion.



Abbildung 86: Das Beispielprogramm zur Ermittlung der Papierkorbinfos in Aktion

151 Verknüpfungen anlegen

Wie Sie ja wahrscheinlich schon wissen, erlaubt Windows das Anlegen von Verknüpfungen (Links) zu Dateien und Ordnern. Verknüpfungen verwalten intern einen Pfad zur Datei bzw. zum Ordner, eventuelle Aufrufargumente (bei Programmdateien) und ein Arbeitsverzeichnis. Über Verknüpfungen, die in einem beliebigen Ordner gespeichert sein können, können Sie Dateien und Ordner öffnen, als würden Sie diese direkt öffnen.

Im Explorer erzeugen Sie eine Verknüpfung zum einen über das Ziehen einer Datei oder eines Ordners mit anschließendem Fallenlassen unter der Betätigung der STRG- und der SHIFT-Taste. Zum anderen können Sie eine Verknüpfung auch über das Explorer-Menü anlegen.

Verknüpfungen können Sie aber (natürlich) auch in einem Programm erzeugen. Sinnvoll ist das z. B., wenn eine Anwendung in bestimmten Situationen automatisch E-Mails versendet, die sich beispielsweise auf Dateien im Netzwerk oder auf bestimmte Datensätze in einer Datenbank beziehen. Über eine Verknüpfung zu den Dateien bzw. zu einem Programm, das die Datensätze anzeigen kann, können Sie dem Anwender die Möglichkeit geben, die assoziierten Informationen auf eine einfache Weise zu öffnen.

Leider bietet das .NET Framework scheinbar auch in der Version 3.5 keine Möglichkeit, Verknüpfungen anzulegen. Dazu müssen Sie wieder einmal das Windows-API bemühen. Etwas problematisch daran ist, dass die Features zur Erzeugung von Verknüpfungen nicht in einer klassischen DLL, sondern in COM-Schnittstellen implementiert sind (die allerdings in diesem Fall funktionsfähige Klassen sind). Die Haupt-Schnittstellen werden mit `IShellLinkA` (ASCII) und `IShellLinkW` (Unicode) bezeichnet. Daneben existiert (u. a.) noch eine Schnittstelle `IPersistFile` zum Lesen und Schreiben von (Verknüpfungs-)Dateien.

Im Internet kursiert dazu eine COM-Typbibliothek *ShellLink.tlb*, in der diese Schnittstellen deklariert sind. (COM-)Anwendungen können diese Typbibliothek referenzieren, um Verknüpfungen anzulegen. In einem .NET-Programm können Sie diese Typbibliothek ebenfalls zur Erzeugung von Verknüpfungen verwenden.

Eine reine .NET-Lösung ist aber immer die bessere. Und die ist möglich, denn Sie können mit speziellen Attributen auch COM-Schnittstellen in ein .NET-Programm importieren. Das Ganze ist leider etwas kompliziert und erfordert eine Menge an COM-Grundlagenwissen.

Erfreulicherweise hat uns diese komplexe Arbeit *Mattias Sjögren* in Form der Dateien *ShellLinkNative.cs* (native Shell-Link-Klassen) und *ShellShortcut.cs* (.NET-Wrapper für die nativen Klassen) abgenommen. Mit Hilfe der Klasse `ShellShortcut` ist es sehr einfach, eine Verknüpfung anzulegen. An der Adresse *www.msjogren.net/dotnet/eng/samples/dotnet_shelllink.asp* finden Sie ein Beispiel, das diese Klassen einsetzt. Die Klassen finden Sie auch im Komponenten-Ordner auf der CD.

Ich stelle die (relativ umfangreichen) Shell-Link-Klassen hier nicht vor, sondern zeige lediglich, wie Sie mit deren Hilfe eine Verknüpfung anlegen. Dazu erzeugen Sie zunächst eine Instanz der Klasse `ShellShortcut`, der Sie im Konstruktor den vollen Dateinamen der zu erzeugenden Verknüpfung übergeben. Die Dateiendung solcher Dateien ist üblicherweise *.lnk*.

Danach stellen Sie zumindest die Eigenschaften `Path` (Pfad zur verknüpften Datei bzw. zum verknüpften Ordner) ein. Handelt es sich bei dieser Datei um ein Programm, können Sie in der Eigenschaft `Arguments` Aufrufargumente definieren. Im Beispiel setze ich den Pfad zum Windows-Editor und übergebe in den Argumenten einen Pfad zu einer Textdatei.

Die Eigenschaft `Description` verwaltet die Beschreibung der Verknüpfung. In der Eigenschaft `WorkingDirectory` können Sie das Arbeitsverzeichnis definieren, das ein Programm vorzugsweise zum Öffnen und Speichern von Dateien verwendet. Über die Eigenschaft `Icon` können Sie die Verknüpfung mit einem Icon versehen, das Sie im Programm selbst erzeugt oder ausgelesen haben. Normalerweise reicht jedoch die Angabe des Pfades zu einer Datei, die das darzustellende

Icon speichert (gewöhnlich ist das die verknüpfte Datei selbst), in der Eigenschaft `IconPath`. Dann müssen Sie lediglich noch den Index des Icons angeben (Programmdateien können mehrere Icons verwalten). Wenn Sie das erste Icon verwenden wollen, geben Sie hier 0 an.

Der folgende Programmcode zeigt die Anwendung der `ShellShortcut`-Klasse am Beispiel einer Verknüpfung zum Windows-Editor, der die Textdatei *eula.txt* im Windows-Systemordner anzeigen soll. Das Programm erfordert die Einbindung der Dateien *ShellLinkNative.cs* und *ShellShortcut.cs* und den Import der Namensräume `System`, `System.IO`, `System.Diagnostics` und `MSjogren.Samples.ShellLink`.

```
// Name der zu erzeugenden Verknüpfungsdatei
string linkFileName = "c:\\eula.lnk";

// Pfad zu notepad.exe zusammenstellen
string path = Path.Combine(Environment.SystemDirectory, "notepad.exe");

// Argumente für den Aufruf
string arguments = Path.Combine(Environment.SystemDirectory, "eula.txt");

// Instanz der Klasse ShellShortcut erzeugen
ShellShortcut ssh = new ShellShortcut(linkFileName);

// ShellShortcut-Objekt initialisieren
ssh.Description = "Test-Verknüpfung"; // Beschreibung
ssh.WorkingDirectory = "c:\\";        // Arbeitsverzeichnis
ssh.Path = path;                      // Pfad zur Datei / zum Ordner
ssh.Arguments = arguments;            // Aufruf-Argumente
ssh.IconPath = linkFileName;          // Pfad zur Icon-Datei
ssh.IconIndex = 0;                    // Index des Icons

try
{
   // Verknüpfung speichern
   ssh.Save();

   Console.WriteLine("Fertig");
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Anlegen der Verknüpfung: {0}",
      ex.Message);
}
```

Listing 259: Erzeugen einer Verknüpfung über die Klasse ShellShortcut von Mattias Sjögren

Die `Save`-Methode speichert die Verknüpfung auch dann, wenn bereits eine gleichnamige Datei existiert. Für den Fall, dass Sie eine bereits vorhandene Verknüpfung nicht überschreiben wollen, sollten Sie also vor dem Anlegen der Verknüpfung überprüfen, ob bereits eine solche

Datei existiert. Da beim Speichern natürlich auch Fehler auftreten können, sollten Sie, wie im Beispiel, die `Save`-Methode innerhalb einer Ausnahmebehandlung aufrufen.

152 Das Dateisystem überwachen

Das Dateisystem daraufhin zu überwachen, dass Dateien oder Ordner erstellt, geändert, umbenannt oder gelöscht werden, ist über eine Instanz der `FileSystemWatcher`-Klasse sehr einfach.

Beim Erzeugen übergeben Sie der `FileSystemWatcher`-Instanz den zu überwachenden Ordner am ersten Argument (oder Sie definieren diesen über die `Path`-Eigenschaft). Die Eigenschaft `IncludeSubdirectories` gibt an, ob auch alle Unterverzeichnisse überwacht werden sollen. Am zweiten Argument des Konstruktors (oder in der Eigenschaft `Filter`) können Sie einen Filter übergeben, der die überwachten Elemente einschränkt. Hier geben Sie normale Dateimuster an, wie zum Beispiel **.txt* für alle Dateien mit der Endung *.txt*. Leider können Sie nicht mehrere Dateimuster kombinieren. Wenn Sie zum Beispiel alle Bilddateien überwachen wollen, müssen Sie auf den Filter verzichten und die Dateien in den Ereignissen der `FileSystemWatcher`-Klasse an Hand der Endung selbst filtern.

`FileSystemWatcher` ist übrigens eine Komponente, die auf ein Windows.Forms-Formular (leider nicht auf ein WPF-Fenster) gezogen werden kann. Das erleichtert das Erzeugen und das Zuweisen der Ereignisse.

Zur Überwachung stehen fünf Ereignisse zur Verfügung, die in Tabelle 16 beschrieben sind.

Ereignis	Bedeutung
`Changed`	Eine Datei oder ein Ordner wurde geändert
`Created`	Eine Datei oder ein Ordner wurde erzeugt. Dieses Ereignis wird auch aufgerufen, wenn eine Datei oder ein Ordner in den überwachten Pfad kopiert oder verschoben wird.
`Deleted`	Eine Datei oder ein Ordner wurde gelöscht
`Renamed`	Eine Datei oder ein Ordner wurde umbenannt
`Error`	wird aufgerufen, wenn bei der Überwachung ein Fehler auftritt. Dies ist z. B. dann der Fall, wenn zu viele Ereignisse einen Überlauf des internen Puffers bewirken. Sie können die Puffergröße über die Eigenschaft `InternalBufferSize` verändern.

Tabelle 16: Die Überwachungsereignisse der FileSystemWatcher-Komponente

Hinweis

Der angegebene Ordner wird nur dann überwacht, wenn die Eigenschaft `EnableRaisingEvents` `true` ist. Die Voreinstellung dieser Eigenschaft ist `false`.

Über die Eigenschaft `FullPath` des den Ereignissen übergebenen Ereignisargument-Objekts erhalten Sie den vollen Pfad der Datei bzw. des Ordners. Die Eigenschaft `Name` liefert lediglich den Datei- oder Ordnernamen. Im Ereignis `Renamed` liefern `FullPath` den neuen Pfad und `Name` den neuen Namen. Über die zusätzlichen Eigenschaften `OldFullPath` und `OldName` erhalten Sie in diesem Ereignis den alten Pfad und den alten Namen. Leider bietet das Ereignis `Created`, das

auch aufgerufen wird, wenn Dateien oder Ordner in den überwachten Ordner kopiert oder verschoben werden, nicht die Möglichkeit, herauszufinden, woher die verschobene bzw. kopierte Datei kam.

Die Methode `WaitForChanged` erlaubt darüber hinaus das Warten auf eine bestimmte Änderung, optional mit einem angegebenen Timeout.

Das folgende Konsolenanwendungs-Beispiel nutzt die Ereignisse einer `FileSystemWatcher`-Instanz zur Überwachung aller möglichen Änderungen im Dateisystem und schreibt eine entsprechende Meldung an die Konsole. Das Beispiel nutzt die Tatsache, dass die Signatur des Delegaten der Ereignisse `Created`, `Changed` und `Deleted` gleich ist und dass am Argument `e.ChangeType` die Art der Änderung übergeben wird.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.IO` einbinden.

```
static void Main(string[] args)
{
   // FileSystemWatcher zum Überwachen des Ordners C:\ erzeugen
   FileSystemWatcher fileSystemWatcher = new FileSystemWatcher("C:\\");

   // Die Ereignisse zuweisen
   fileSystemWatcher.Created += new FileSystemEventHandler(
      fileSystemWatcher_CreatedOrChangedOrDeleted);
   fileSystemWatcher.Changed += new FileSystemEventHandler(
      fileSystemWatcher_CreatedOrChangedOrDeleted);
   fileSystemWatcher.Deleted += new FileSystemEventHandler(
      fileSystemWatcher_CreatedOrChangedOrDeleted);
   fileSystemWatcher.Renamed += new RenamedEventHandler(
      fileSystemWatcher_Renamed);
   fileSystemWatcher.Error += new ErrorEventHandler(
      fileSystemWatcher_Error);

   // Die Ereignisüberwachung einschalten
   fileSystemWatcher.EnableRaisingEvents = true;

   Console.WriteLine("Der Ordner C:\\ wird überwacht ...");

   Console.WriteLine();
   Console.WriteLine("Beenden mit Return");
   Console.ReadLine();
}

static void fileSystemWatcher_CreatedOrChangedOrDeleted(object sender,
   FileSystemEventArgs e)
{
```

Listing 260: Konsolenanwendungs-Beispiel zum Überwachen des Dateisystems

```
   switch (e.ChangeType)
   {
      case WatcherChangeTypes.Changed:
         Console.WriteLine(e.FullPath + " wurde geändert");
         break;

      case WatcherChangeTypes.Created:
         Console.WriteLine(e.FullPath + " wurde erstellt");
         break;

      case WatcherChangeTypes.Deleted:
         Console.WriteLine(e.FullPath + " wurde gelöscht");
         break;
   }
}

static void fileSystemWatcher_Renamed(object sender, RenamedEventArgs e)
{
   Console.WriteLine(e.OldFullPath + " wurde umbenannt in " + e.FullPath);
}

static void fileSystemWatcher_Error(object sender, ErrorEventArgs e)
{
   Console.WriteLine("Fehler beim Überwachen des Dateisystems: " +
      e.GetException().Message);
}
```

Listing 260: Konsolenanwendungs-Beispiel zum Überwachen des Dateisystems (Forts.)

Text-, binäre und Zip-Dateien

153 Textdateien lesen

Textdateien können Sie auf eine sehr einfache Weise über eine Instanz der Klasse `System.IO.StreamReader` lesen. Die Methode `Read` erlaubt das Lesen einzelner Zeichen, über die Methode `ReadLine` können Sie zeilenweise lesen, die Methode `ReadToEnd` ermöglicht das Einlesen einer Textdatei in einem Rutsch. `Read` und `ReadLine` geben einen String zurück, wenn (noch) gelesen werden konnte. Ist der Stream am Ende der Datei angelangt, geben beide Methoden `null` zurück. In einer Schleife, die den jeweils eingelesenen String auf ungleich `null` vergleicht, können Sie sehr einfach bis zum Ende der Datei einlesen.

Eine `StreamReader`-Instanz können Sie über die Methode `OpenText` eines `FileInfo`-Objekts erzeugen. Den Dateinamen übergeben Sie am ersten Argument des Konstruktors:

```
System.IO.FileInfo fi = new System.IO.FileInfo(fileName);
System.IO.StreamReader sr = fi.OpenText();
```

Alternativ können Sie die gleichnamige statische Methode der `File`-Klasse verwenden:

```
System.IO.StreamReader sr = System.IO.File.OpenText(fileName);
```

Beide Varianten besitzen den Nachteil, dass Sie die Codierung der Datei nicht angeben können. Textdateien können ja, wie Sie wahrscheinlich bereits wissen, in verschiedenen ASCII- oder Unicode-Codierungen gespeichert sein. Die Methode `OpenText` liest die Datei in beiden Varianten mit der UTF-8-Codierung ein. Das führt spätestens dann zu inkorrekt dargestellten Texten, wenn der Text Sonderzeichen (wie unsere Umlaute) enthält und in einer anderen Codierung gespeichert ist.

Erzeugen Sie die `StreamReader`-Instanz stattdessen explizit, können Sie die Codierung selbst bestimmen. Dazu übergeben Sie am zweiten Argument eine Instanz der Klasse `System.Text.Encoding`. Diese können Sie über einige statische Eigenschaften der `Encoding`-Klasse (die `Encoding`-Instanzen für Standard-Codierungen zurückgeben) oder über die Methode `GetEncoding` (der Sie den Namen oder die Id einer Codierung übergeben) erzeugen.

Eigenschaft/Methode	Bedeutung
`ASCII`	7-Bit (!) ASCII-Codierung
`BigEndianUnicode`	Unicode in der Byte-Reihenfolge *Big-Endian*. Intel-Plattformen verwenden standardmäßig die Reihenfolge *Little Endian*.
`Default`	die vom System verwendete Standard-Codierung
`Unicode`	Standard-Unicode
`UTF7`	Codierung im Format UTF-7, bei dem Unicode-Daten so konvertiert werden, dass diese einem 7-Bit-ASCII-Code entsprechen
`UTF8`	Codierung im Format UTF-8, bei dem Unicode-Daten so konvertiert werden, dass diese einem 8-Bit-ASCII-Code entsprechen

Tabelle 17: Die statischen Eigenschaften der Encoding-Klasse zur Ermittlung einer Standard-Codierung

Voraussetzung dafür ist natürlich, dass Sie die Codierung der Textdatei kennen, was in der Praxis oft nicht allzu einfach herauszufinden ist.

Das folgende Beispiel erzeugt eine `StreamReader`-Instanz für die Codierung *Windows-1252* (Windows-Standard):

```
System.IO.StreamReader sr = new System.IO.StreamReader(fileName,
   System.Text.Encoding.GetEncoding("windows-1252"));
```


Kennen Sie die Codierung nicht, können Sie über `true` am zweiten Argument erreichen, dass die `GetEncoding`-Methode versucht, die Codierung automatisch an den ersten Bytes der Textdatei zu erkennen:

```
System.IO.StreamReader sr = new System.IO.StreamReader(fileName, true);
```

Diese automatische Erkennung funktioniert allerdings nur für Dateien im UFT-8- oder in einem Unicode-Format, und auch nur dann, wenn diese mit einer so genannten Byte-Reihenfolgemarkierung beginnen. In der Praxis können Sie mit dieser Variante also wenig anfangen, besonders dann, wenn Sie 8-Bit-Textdateien einlesen.

Das eigentliche Lesen ist dann sehr einfach. Da in den meisten Fällen wohl zeilenweise gelesen wird, zeige ich diese Variante. Das Beispiel in Listing 261 liest die Datei *hitchhiker.txt* ein und gibt die gelesenen Zeilen an der Konsole aus. Die Datei wird im Ordner der Anwendung erwartet. Da gerade beim Öffnen von Dateien Ausnahmen des Typs `IOException` möglich sind (z. B., weil die Datei nicht vorhanden oder gerade exklusiv gesperrt ist), werden diese abgefangen.

Das Beispiel benötigt die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms`.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "hitchhiker.txt");

// StreamReader-Instanz für die Datei erzeugen
StreamReader sr = null;
try
{
   sr = new StreamReader(fileName, Encoding.GetEncoding("windows-1252"));
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
      ex.Message, Application.ProductName, MessageBoxButtons.OK,
   MessageBoxIcon.Error);
   return;
}

// Datei zeilenweise einlesen
string line = null;
while ((line = sr.ReadLine()) != null)
```

Listing 261: Zeilenweises Einlesen einer Textdatei mit der Codierung Windows-1252

```
{
   Console.WriteLine(line);
}

// StreamReader schließen
sr.Close();
```

Listing 261: Zeilenweises Einlesen einer Textdatei mit der Codierung Windows-1252 (Forts.)

Zum Kompilieren dieses Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System.IO`, `System.Text`, `System.Reflection` und `System.Windows.Forms` importieren.

Abbildung 87 zeigt das ausgeführte Programm.

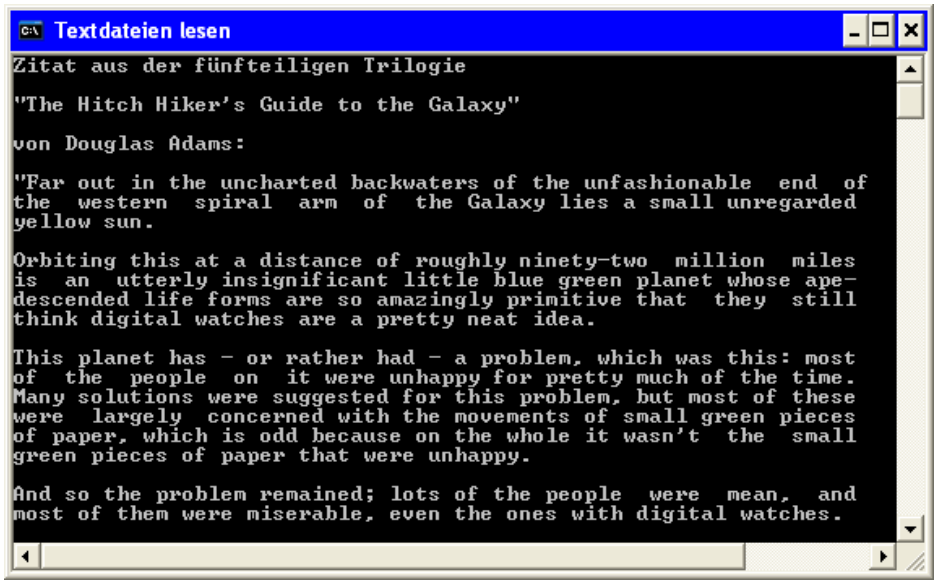


Abbildung 87: Das Beispielprogramm hat die Textdatei eingelesen

154 Textdateien schreiben

Das Schreiben von Textdateien ist ähnlich einfach wie das Lesen (siehe vorhergehendes Rezept). Das einzige Problem ist wieder die Einstellung der korrekten Codierung.

Zum Erstellen einer Textdatei erzeugen Sie eine Instanz der Klasse `StreamWriter` (aus dem Namensraum `System.IO`). Deren Konstruktor existiert in verschiedenen überladenen Varianten. So können Sie einen Dateinamen am ersten Argument des Konstruktors angeben, womit die `StreamWriter`-Instanz implizit einen Stream zum Schreiben der Datei erzeugt. Alternativ können Sie aber auch ein existierendes `Stream`-Objekt (oder eine Instanz einer von `Stream` abgeleiteten Klasse) angeben, um zu erreichen, dass das `StreamWriter`-Objekt in dieses schreibt. In weiteren Argumenten können Sie festlegen, ob an vorhandene Dateien angehängt werden soll,

Sie können die Codierung bestimmen etc. Ich verwende im Beispiel die Variante, die einen Dateinamen am ersten Argument erwartet.

Bei dieser Variante übergeben Sie am ersten Argument also den Namen der Textdatei. Am zweiten Argument können Sie festlegen, ob die Textdaten an eine eventuell bereits vorhandene Datei angehängt werden oder ob diese überschrieben wird. Das dritte Argument bestimmt die Codierung der Textdatei.

Über die Methode `Write` können Sie dann einzelne Zeichenketten in die Datei schreiben. Die Methode `WriteLine` schreibt eine Zeichenkette und hängt zusätzlich die für das System gültigen Zeilenendezeichen (unter Windows sind das die Zeichen "\r\n") an diese an.

Wenn Sie neue Textdateien schreiben (also beim Erzeugen der `StreamWriter`-Instanz am zweiten Argument `false` übergeben), sollten Sie beachten, dass ein `StreamWriter` vorhandene Dateien ohne Warnung überschreibt. Sie sollten also gegebenenfalls vor dem Schreiben überprüfen, ob eine gleichnamige Datei bereits existiert.

Das folgende Beispiel erzeugt eine neue Textdatei *fish.txt* im Ordner der Anwendung mit der Codierung *Windows-1252* und beschreibt diese mit zwei Zeilen Text. Ist die zu erzeugende Datei bereits vorhanden, fragt das Programm den Anwender, ob er diese überschreiben will. Ausnahmen vom Typ `IOException`, die im Besonderen beim Öffnen der Datei erzeugt werden können, werden abgefangen.

Um das Beispiel kompilieren zu können müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text`, `System.Reflection` und `System.Windows.Forms` importieren.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "fish.txt");

// Überprüfen, ob die Datei bereits existiert
if (File.Exists(fileName))
{
   // Den Anwender fragen, ob er die Datei überschreiben will
   if (MessageBox.Show("Die Datei '" + fileName + "' existiert " +
      "bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
      Application.ProductName, MessageBoxButtons.YesNo,
      MessageBoxIcon.Question) == DialogResult.No)
      return;
}

// StreamWriter-Instanz für die Datei erzeugen
StreamWriter sw = null;
try
{
   sw = new StreamWriter(fileName, false,
      Encoding.GetEncoding("windows-1252"));
```

Listing 262: Schreiben einer Textdatei in der Codierung Windows-1252

```
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
      ex.Message, Application.ProductName, MessageBoxButtons.OK,
      MessageBoxIcon.Error);
   return;
}

// Datei zeilenweise schreiben
sw.WriteLine("Macht's gut");
sw.WriteLine("und danke für den Fisch");

// StreamWriter schließen
sw.Close();
```

Listing 262: Schreiben einer Textdatei in der Codierung Windows-1252 (Forts.)

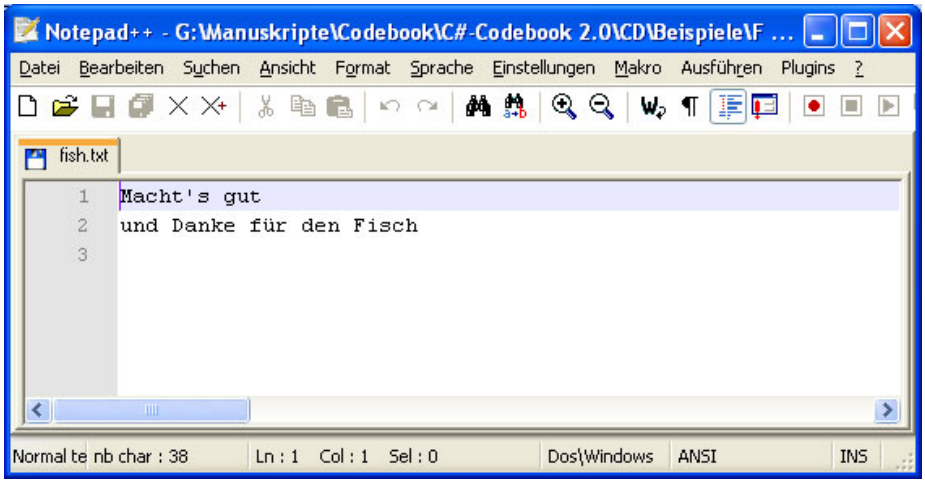


Abbildung 88: Die erzeugte Textdatei in einem Editor

155 Texte an Textdateien anfügen

Das Anfügen von Texten an vorhandene Textdateien ist wie das Schreiben neuer Dateien sehr einfach. Dazu erzeugen Sie wie im vorhergehenden Rezept eine Instanz der Klasse `StreamWriter`, übergeben aber am zweiten Argument `append` nun `true`. Damit wird eine vorhandene Datei so geöffnet, dass neue Zeichen oder Zeichenketten an das Ende angefügt werden. Ist die Datei noch nicht vorhanden, wird sie automatisch angelegt.

Das Beispiel in Listing 263 hängt einen String mit dem aktuellen Datum und der aktuellen Anzahl ausgeführter Prozesse an die Datei *log.txt* an. Diese Datei wird im Ordner der Anwendung erwartet. Ausnahmen, die im Besonderen beim Öffnen der Datei entstehen können, werden abgefangen.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text` und `System.Windows.Forms` importieren.

```
// Dateiname ermitteln
string fileName = Path.Combine(Application.StartupPath, "log.txt");

// StreamWriter-Instanz für die Datei erzeugen
StreamWriter sw = null;
try
{
   sw = new StreamWriter(fileName, true, Encoding.GetEncoding(
      "windows-1252"));
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Öffnen der Datei '" + fileName + "': " +
      ex.Message, Application.ProductName, MessageBoxButtons.OK,
      MessageBoxIcon.Error);
   return;
}

// Zeile anfügen
sw.WriteLine(DateTime.Now.ToString() + ": Anzahl Prozesse: " +
   System.Diagnostics.Process.GetProcesses().Length);

// StreamWriter schließen
sw.Close();
```

Listing 263: Anhängen von Texten an eine Textdatei

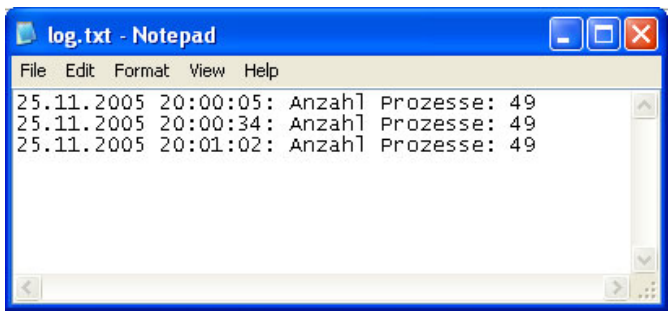


Abbildung 89: Die vom Beispielprogramm nach drei Ausführungen erzeugte Textdatei in einem Editor

156 Textdateien ändern

Das Ändern von Textdateien ist nicht direkt möglich. Dazu müssen Sie die Datei einlesen, im Arbeitsspeicher ändern, die Originaldatei löschen und schließlich neu schreiben. Etwas problematisch wird dies, wenn die Datei sehr groß ist und folglich den Arbeitsspeicher beim kompletten Einlesen überlasten würde. In diesem Fall können Sie so vorgehen, dass Sie die Datei Zeile für Zeile einlesen, die einzelnen Zeilen bearbeiten und die gewünschten (und eventuell veränderten Zeilen) in eine temporäre Datei schreiben. Nach dem Bearbeiten löschen Sie die originale Textdatei und verschieben die temporäre Datei mit dem Namen der Textdatei in den Ordner derselben.

Listing 264 implementiert dieses zeilenweise Verändern einer Textdatei in Form der Methode `AddFileNumbers`, die der Datei Zeilennummern hinzufügt.

Zum Kompilieren der Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Text` importieren.

```
public static void AddFileNumbers(string textFileName)
{
   // Dateiname für die temporäre Datei ermitteln
   string tempFileName = Path.GetTempFileName();

   // Textdatei zum Lesen öffnen
   StreamReader sr = null;
   sr = new StreamReader(textFileName, Encoding.GetEncoding("windows-1252"));

   // Temp-Datei zum Schreiben öffnen
   StreamWriter sw = null;
   sw = new StreamWriter(tempFileName, false,
      Encoding.GetEncoding("windows-1252"));

   // Textdatei zeilenweise einlesen, Zeilen verändern und in die temporäre
   // Datei schreiben
   string line;
   int lineNumber = 0;
   while ((line = sr.ReadLine()) != null)
   {
      lineNumber++;
      sw.WriteLine(lineNumber + ": " + line);
   }

   // Streams schließen
   sr.Close();
   sw.Close();
```

Listing 264: Verändern einer Textdatei

```
    // Textdatei löschen und die temporäre Datei auf die Textdatei verschieben
    File.Delete(textFileName);
    File.Move(tempFileName, textFileName);
}
```

Listing 264: Verändern einer Textdatei (Forts.)

157 Binäre Dateien lesen

Binäre Dateien können Sie über eine Instanz der Klasse `FileStream` (aus dem Namensraum `System.IO`) einlesen. Ähnlich dem Lesen von Textdateien können Sie diese Instanz über die Methode `OpenRead` einer Instanz der Klasse `FileInfo` oder der `File`-Klasse erzeugen:

```
System.IO.FileStream fs = System.IO.File.OpenRead(fileName);

...

System.IO.FileInfo fi = new System.IO.FileInfo(fileName);
System.IO.FileStream fs = fi.OpenRead();
```

Alternativ (aber hier nicht unbedingt vorteilhaft) können Sie die Instanz der `FileStream`-Klasse auch selbst erzeugen, wobei Sie am zweiten Argument den Modus zum Öffnen (`FileMode.Open`) und am dritten Argument die Zugriffsart (`FileAccess.Read`) übergeben:

```
System.IO.FileStream fs = new System.IO.FileStream(fileName,
    FileMode.Open, FileAccess.Read)
```

Lesen können Sie die Daten dann über die `Read`-Methode. Dieser Methode übergeben Sie am ersten Argument ein `byte`-Array, das Sie zuvor ausreichend groß dimensionieren müssen. In dieses Array schreibt die Methode die gelesenen Bytes.

Byte-Arrays werden unter .NET vorzugsweise für binäre Daten verwendet. Den meisten Streams können Sie im Konstruktor unter anderem ein solches Array übergeben, um die darin gespeicherten Daten an einen anderen Ort zu übertragen.

Am zweiten Argument übergeben Sie die Position, ab der im Stream gelesen werden soll. In der Praxis übergeben Sie hier eigentlich immer 0 um ab der aktuellen Position zu lesen. Das dritte Argument bestimmt die Anzahl der zu lesenden Bytes. Dieses Argument ist wichtig, wenn Sie, wie weiter unten beschrieben, Daten blockweise einlesen.

Kleinere Dateien können Sie in einem Rutsch lesen, wie es der Programmcode in Listing 265 zeigt. Dazu müssen Sie allerdings wissen, wie groß die einzulesende Datei ist, was Sie über die `Length`-Eigenschaft eines `FileInfo`-Objekts erfahren können. Das Programm liest die Datei *hitchhiker.jpg* ein und speichert diese Base-64-codiert[1] in eine XML-Datei *hitchhiker.xml*. Beide Dateien werden im Ordner der Anwendung erwartet.

Das Beispiel erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Xml`, `System.Diagnostics` und `System.Windows.Forms`.

1. Eine Base-64-Codierung bewirkt, dass binäre Daten so umgesetzt werden, dass diese einem 7-Bit-ASCII-Code entsprechen und problemlos in Textdateien gespeichert oder über das Internet übertragen werden können.

```
// Dateinamen ermitteln
string filename = Path.Combine(Application.StartupPath, "hitchhiker.jpg");
string xmlFileName = Path.Combine(Application.StartupPath, "hitchhiker.xml");

// FileStream-Instanz erzeugen
FileInfo fi = new FileInfo(filename);
FileStream fs = fi.OpenRead();

// Byte-Puffer erzeugen und die Datei in diesen einlesen
byte[] buffer = new Byte[fi.Length];
fs.Read(buffer, 0, buffer.Length);

// FileStream schließen
fs.Close();

// Die eingelesenen Daten Base-64-codiert in eine XML-Datei schreiben
XmlTextWriter xmlWriter = new XmlTextWriter(xmlFileName, Encoding.UTF8);
xmlWriter.WriteStartDocument(true);
xmlWriter.WriteStartElement("applicationData");
xmlWriter.WriteStartElement("splashImage");
xmlWriter.WriteBase64(buffer, 0, buffer.Length);
xmlWriter.WriteEndElement();
xmlWriter.WriteEndElement();

// XML-Datei schließen
xmlWriter.Close();

// XML-Datei anzeigen
Process.Start(xmlFileName);
```

Listing 265: Einlesen einer binären Datei in einem Rutsch in eine XML-Datei

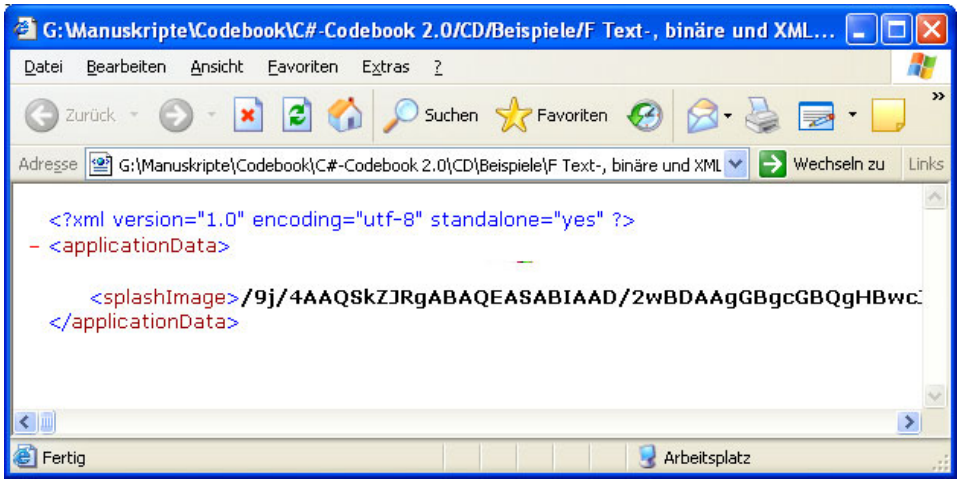


Abbildung 90: Die erzeugte XML-Datei im Internet Explorer

Das Beispiel ist übrigens nicht allzu weit von der Praxis entfernt. Die Speicherung binärer Daten in einer XML-Datei erlaubt es Ihnen z. B., beliebige Daten (u. a. wie im Beispiel auch Bilder) in einer einzigen Datei zu speichern.

158 Binäre Dateien schreiben

Das Schreiben binärer Daten ist ähnlich einfach wie das Lesen (im vorhergehenden Rezept). Dazu erzeugen Sie ein `FileStream`-Objekt (aus dem Namensraum `System.IO`) und rufen dessen `Write`-Methode auf. Dieser Methode übergeben Sie ein `byte`-Array, eine Startposition für den Stream (normalerweise 0) und die Anzahl der zu schreibenden Bytes. Das `FileStream`-Objekt können Sie über die Methode `OpenWrite` der `FileInfo`-Klasse oder einer Instanz der `File`-Klasse erzeugen. Natürlich ist auch das direkte Erzeugen möglich, wobei Sie im Konstruktor neben dem Dateinamen mindestens den Datei-Modus (`FileAccess.Write`) übergeben müssen. Ideal ist eigentlich die Verwendung der `File`-Klasse, da Sie hier nur den Dateinamen angeben müssen.

Das folgende Beispiel liest die im vorhergehenden Rezept erzeugten XML-Daten ein und erzeugt daraus eine JPEG-Grafikdatei. Da unbekannt ist, wie groß die zu erzeugende Datei wird, werden die Daten blockweise in Stücken von einem KB eingelesen.

Zum Kompilieren dieses Programms müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text`, `System.Xml` und `System.Windows.Forms` importieren.

```
// Dateinamen ermitteln
string xmlFileName = Path.Combine(Application.StartupPath, "hitchhiker.xml");
string imageFileName = Path.Combine(Application.StartupPath,
   "hitchhiker.jpg");

// FileStream-Instanz zum Schreiben erzeugen
FileStream fs = File.OpenWrite(imageFileName);

// XML-Datei öffnen
XmlTextReader xmlReader = new XmlTextReader(xmlFileName);

// Zum splashImage-Element navigieren (das dem applicationData-Element folgt)
xmlReader.ReadStartElement("applicationData");

// XML-Daten des splashImage-Elements blockweise einlesen
byte[] buffer = new Byte[1024];
int count = 0;
do
{
   count = xmlReader.ReadBase64(buffer, 0, buffer.Length);
   if (count > 0)
   {
```

Listing 266: Erzeugen einer binären Datei aus den Base64-Daten einer XML-Datei

```
      // Eingelesene Daten in den FileStream schreiben
      fs.Write(buffer, 0, count);
   }
} while (count > 0);

// XML-Datei schließen
xmlReader.Close();

// FileStream schließen
fs.Close();

// Das erzeugte Bild anzeigen
Process.Start(imageFileName);
```

Listing 266: Erzeugen einer binären Datei aus den Base64-Daten einer XML-Datei (Forts.)

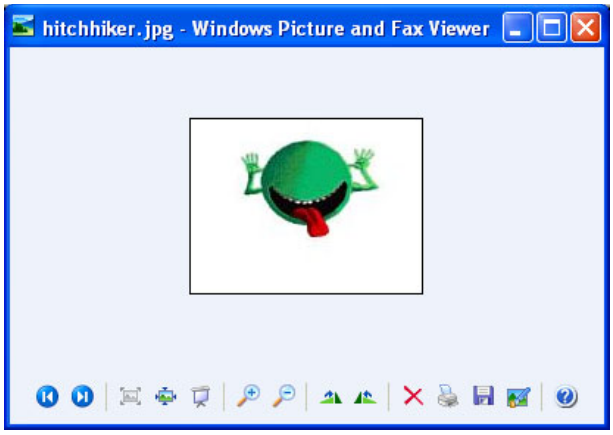


Abbildung 91: Das aus den XML-Daten erzeugte JPEG-Bild

159 Binäre Dateien Base64-codieren

In vielen Fällen ist es notwendig, binäre Dateien so zu codieren, dass diese problemlos über das Internet versendet werden können. Dazu bietet sich eine Base64-Codierung an, die im Internet weit verbreitet ist. Base64 codiert binäre Daten so, dass diese dem 7-Bit-ASCII-Code entsprechen.

Die Methode `ReadAsBase64` in Listing 267 erzeugt aus einer Datei einen Base64-codierten String. Die Datei wird dazu zunächst über eine Instanz der Klasse `FileStream` in ein `byte`-Array eingelesen. Dieses Array übergibt die Methode an die Methode `ToBase64String` der `Convert`-Klasse, die die übergebenen Bytes Base64-codiert.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.IO` importieren.

```
public static string ReadAsBase64(string fileName)
{
   // Datei einlesen
   FileInfo fi = new FileInfo(fileName);
   byte[] buffer = new byte[fi.Length];
   FileStream fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);
   fs.Read(buffer, 0, buffer.Length);
   fs.Close();

   // Das Byte-Array Base64-codiert zurückgeben
   return Convert.ToBase64String(buffer, 0, buffer.Length);
}
```

Listing 267: Methode zum Lesen einer Datei als Base64-codierten String



Abbildung 92: Beispielprogramm, das eine JPEG-Datei eingelesen und nach Base64 codiert hat

160 Base64-codierte Strings in Dateien umwandeln

Im Rezept 159 habe ich eine Methode vorgestellt, die eine binäre Datei in einen Base64-codierten String umwandelt. Die Methode `CreateFileFromBase64` in Listing 268 macht genau das Gegenteil: die Umwandlung eines Base64-codierten Strings in ein Byte-Array, das dann als Datei gespeichert wird. Zur Umwandlung setzt `CreateFileFromBase64` die Methode `FromBase64String` der `Convert`-Klasse ein.

Um diese Methode kompilieren zu können, müssen Sie die Namensräume `System` und `System.IO` importieren.

```
public static void CreateFileFromBase64(string base64CodedString,
   string filePath)
{
   // String in Byte-Array konvertieren
   byte[] buffer = Convert.FromBase64String(base64CodedString);

   // Datei über einen FileStream schreiben
   FileStream fs = new FileStream(filePath, FileMode.Create,
      FileAccess.Write);
   fs.Write(buffer, 0, buffer.Length);
   fs.Close();
}
```

Listing 268: Methode zum Erzeugen einer Datei aus einem Base64-codierten String

161 Komprimieren von Daten mit .NET-Klassen

Das .NET Framework bietet insgesamt drei Klassen zum Komprimieren (und Dekomprimieren) von Daten:

- `System.IO.Compression.GZipStream`: Verwendet den GZIP-Algorithmus. Dieser Algorithmus ist im Vergleich zu Deflate besser geeignet, wenn die Daten in einer Datei gespeichert werden sollen, weil der GZIP-Algorithmus (in Form von Dateien mit der Endung *.gz*) weiter verbreitet ist.
- `System.IO.Compression.DeflateStream`: Verwendet den Deflate-Algorithmus, der auf demselben Komprimier-Algorithmus basiert wie GZIP. Deflate ist angeblich besser geeignet, wenn die Daten über ein Netzwerk versendet werden.
- `System.IO.Packaging.ZipPackage`: Verwendet den ZIP-Algorithmus und ermöglicht erweiterte Features wie das Speichern mehrerer logischer Dateien.

`GZipStream` und `DeflateStream` sind relativ einfache Klassen, die lediglich die Daten eines Stream nach dem jeweiligen Algorithmus komprimieren bzw. dekomprimieren.

Die Klasse `ZipPackage` (aus dem Namensraum `System.IO.Packaging`, der die Referenzierung der Assembly *Windows.Base.dll* erfordert) arbeitet etwas anders. Da eine `ZipPackage`-Instanz ein ZIP-Archiv repräsentiert, können Sie mehrere logische Dateien (in Form von `ZipPackage`-Objekten) darin verwalten. Außerdem unterstützt `ZipPackage` in einer sehr eingeschränkten Weise die Einstellung des Komprimierungsgrads.

Hinweis

Der Komprimierungsgrad kann laut eigenen Tests und der Dokumentation (siehe *msdn2.microsoft.com/de-de/library/system.io.packaging.package.createpartcore.aspx*) allerdings nur auf die Komprimierungsoptionen `NotCompressed` und `Normal` gesetzt werden. Bei allen anderen Werten (wie z. B. `Maximum`) wird implizit eine normale Kompression verwendet.

Da alle drei Klassen mit Streams arbeiten, können Sie diese sehr flexibel einsetzen, nicht nur zum Packen und Entpacken von Dateien, sondern z. B. auch zum Komprimieren von Daten, die über das Internet versendet werden.

Damit Sie eine Idee von der Performance dieser Klassen erhalten, habe ich einen Performancetest implementiert, den Sie neben dem Beispiel zu diesem Rezept finden. Tabelle 18 stellt das Ergebnis auf meinem (alten) Rechner (AMD 64, 4400+) dar. Der Performancetest integriert auch die Bibliothek *#ziblib* (*www.icsharpcode.net/OpenSource/SharpZipLib*), die ab Rezept 166 behandelt wird. Für *#ziplib* habe ich jeweils die minimale, eine mittlere und die maximal mögliche Kompression gewählt. Gemessen wurde nur die Zeit, die das Komprimieren benötigt, nicht die Zeit, die das Speichern der jeweiligen Dateien in Anspruch nahm.

Klasse	Komprimierung einer Textdatei mit 100 KB Größe		Komprimierung einer Bitmap-Datei mit 900 KB Größe und vielen Farben		Komprimierung einer Bitmap-Datei mit 1001 KB Größe und wenigen Farben	
	Größe	Zeit	Größe	Zeit	Größe	Zeit
`DeflateStream`	44,80 KB	12,89 ms	1042,9 KB	125,23 ms	49,55 KB	36,85 ms
`GZipStream`	44,81 KB	14,31 ms	1042,9 KB	130,70 ms	49,56 KB	46,96 ms
`ZipPackage` mit normaler Kompression	45,34 KB	21,69 ms	1043,0 KB	133,76 ms	56,37 KB	51,57 ms
`#ziplib` mit minimaler Kompression (1 von 9)	28,71 KB	49,75 ms	706,06 KB	141,79 ms	27,53 KB	32,29 ms
`#ziplib` mit mittlerer Kompression (4 von 9)	29,76 KB	13,24 ms	706,89 KB	152,04 ms	27,71 KB	32,05 ms
`#ziplib` mit maximaler Kompression	25,29 KB	32,40 ms	719,65 KB	257,83 ms	22,64 KB	955,0 ms

Tabelle 18: Ein Vergleich der Komprimierung der Klassen DeflateStream, GZipStream und ZipPackage und der Bibliothek #ziblib

Wie Sie dem Testergebnis entnehmen können, sind `DeflateStream` und `GZipStream` nahezu identisch. `ZipPackage` ist zwar langsamer und erzeugt u. U. etwas größere Archive, dafür bietet diese Klasse die Möglichkeit, mehrere logische Dateien in einem Archiv abzulegen und für diese den Dateinamen und den Typ festzulegen.

Hinweis

#ziplib ist mit der mittleren Kompression aber der absolute Gewinner. Mit dem eher geringen Kompressionsgrad 4 benötigt diese Klasse etwa so viel Zeit wie die schnellen Klassen `DeflateStream` und `GZipStream`, erzeugt aber kleinere Archive. #ziplib wird ab Rezept 166 behandelt.

Interessant ist, dass die maximale Kompression zwar (teilweise wesentlich) mehr Zeit benötigt, die Archive aber nicht wesentlich kleiner werden. Bei der Bilddatei mit vielen Farben wird das Archiv sogar größer. Weiterhin interessant ist, dass das Komprimieren der Textdaten in der mittleren Kompression wesentlich schneller ist als in der niedrigen. Eine mittlere Kompression scheint also keine schlechte Wahl zu sein.

Das Prinzip der Verwendung der `DeflateStream`- und der `GZipStream`-Klasse ist Folgendes: Sie erzeugen eine Instanz und übergeben im Konstruktor einen Stream, der beim Komprimieren die gepackten und beim Dekomprimieren die entpackten Daten aufnimmt. Am zweiten Argument des Konstruktors legen Sie mit `CompressionMode.Compress` fest, dass Sie komprimieren, und mit `CompressionMode.Decompress`, dass Sie dekomprimieren wollen. Am optionalen dritten Argument können Sie bestimmen, ob der übergebene Stream automatisch geschlossen wird, wenn Sie die `Close`-Methode des `DeflateStream`- bzw. des `GZipStream`- Objekts aufrufen.

Eine `ZipPackage`-Instanz arbeitet etwas anders, da diese ein ZIP-Archiv repräsentiert. Ein solches enthält eine oder mehrere logische Dateien, die in Form von `ZipPackagePart`-Objekten dargestellt werden. Jedes `ZipPackagePart`-Objekt besitzt einen logische URI (Uniform Resource Identifier) und einen MIME-Typ. »URI« und »MIME« sind Begriffe, die im Internet Verwendung finden. Ein URI ist der logische Name einer Ressource, über den diese identifiziert werden kann. Eine E-Mail-Adresse ist z. B. ein URI, ein Dateiname ebenfalls. Ein MIME-Typ gibt die Art von Daten an. Die MIME-Typen sind standardisiert. `text/plain` steht z. B. für einfachen Text.

`ZipPackagePart`-Objekte erzeugen Sie über die `CreatePart`-Methode der `ZipPackage`-Instanz. Dabei übergeben Sie den URI, den Mime-Typen und (optional) den Kompressionsgrad. Die `GetStream`-Methode eines `ZipPackagePart`-Objekts liefert einen Stream, über den Sie Daten lesen oder schreiben können.

Im ersten Beispiel demonstriere ich die `GZipStream`-Klasse. In diesem Beispiel verwende ich zum Komprimieren von Daten einen `FileStream`, der die Quelldaten aus einer Datei liest. Für die komprimierten Daten setze ich ebenfalls einen `FileStream` ein, der die Daten in eine Datei schreibt. Mit diesen initialisiere ich eine neue Instanz der `GZipStream`-Klasse. Das Beispiel liest die Quelldaten blockweise in ein Byte-Array ein (um bei sehr großen Dateien den Speicher nicht zu überfordern) und schreibt diese in den jeweiligen Stream.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System.IO` und `System.IO.Compression` importieren.

```
string sourceFilename = "C:\\Hitchhiker.txt";
string destFilename = "C:\\TempHitchhiker.gz";

// Quelldatei in den Quellstream lesen
using (FileStream sourceStream = new FileStream(sourceFilename, FileMode.Open))
{
```

Listing 269: Komprimieren von Daten mit der GZipStream-Klasse

```
   // Stream für die komprimierten Daten erzeugen
   using (FileStream resultStream = new FileStream(destFilename,
      FileMode.Create))
   {
      // GZipStream erzeugen
      using (GZipStream gzipStream = new GZipStream(
         resultStream, CompressionMode.Compress))
      {
         // Daten blockweise aus dem Quellstream in den
         // GZipStream lesen
         const int blockSize = 1024 * 1024;
         byte[] buffer = new byte[blockSize];
         int bytesRead;
         while ((bytesRead = sourceStream.Read(buffer, 0, buffer.Length)) > 0)
         {
            gzipStream.Write(buffer, 0, bytesRead);
         }
      }
   }
}
```

Listing 269: Komprimieren von Daten mit der GZipStream-Klasse (Forts.)

Im zweiten Beispiel setze ich die `ZipPackage`-Klasse ein um eine Datei in ein ZIP-Archiv zu komprimieren. Das Beispiel erfordert die Referenzierung der Assembly *WindowsBase.dll* und den Import der Namensräume `System`, `System.IO`, `System.IO.Packaging` und `System.Net.Mime`.

```
string sourceFilename = "C:\\Hitchhiker.txt";
string destFilename = "C:\\Hitchhiker.zip";
string packagePartName = "/Hitchhiker.txt";

// Quelldatei in den Quellstream lesen
using (FileStream sourceStream = new FileStream(sourceFilename, FileMode.Open))
{
   // Stream für das Ergebnis erzeugen
   using (FileStream destStream = new FileStream(destFilename, FileMode.Create))
   {
      // Ein ZipPackage erzeugen
      ZipPackage zipPackage = (ZipPackage)Package.Open(destStream, FileMode.Create);

      // Ein ZipPackagePart erzeugen (eine Datei im Archiv)
      ZipPackagePart zipPackagePart = (ZipPackagePart)zipPackage.CreatePart(
         new Uri(packagePartName, UriKind.Relative),
         MediaTypeNames.Text.Plain, CompressionOption.Normal);
```

Listing 270: Komprimieren von Daten mit der ZipPackage-Klasse

```
      // Den Stream des ZipPackagePart holen
      Stream zipStream = zipPackagePart.GetStream();

      // Daten blockweise aus dem Quellstream in den Zip-Stream lesen
      const int blockSize = 1024 * 1024;
      byte[] buffer = new byte[blockSize];
      int bytesRead;
      while ((bytesRead = sourceStream.Read(buffer, 0, buffer.Length)) > 0)
      {
         zipStream.Write(buffer, 0, bytesRead);
      }

      // Das ZipPackage schließen
      zipPackage.Close();
   }
}
```

Listing 270: Komprimieren von Daten mit der ZipPackage-Klasse (Forts.)

162 Dekomprimieren von Daten mit .NET-Klassen

Zum Dekomprimieren von Daten, die mit dem ZIP-, dem Deflate- oder dem GZIP-Algorithmus komprimiert wurden, können Sie die Klassen `ZipPackage`, `DeflateStream` bzw. `GZipStream` verwenden. Die grundlegende Arbeitsweise dieser Klassen habe ich bereits im Rezept 161 erläutert.

Bei `DeflateStream` und `GZipStream` benötigen Sie zum Dekomprimieren von Daten einen Stream, der die komprimierten Daten aufnimmt. Diesen übergeben Sie am Konstruktor. Am zweiten Argument des Konstruktors geben Sie mit `CompressionMode.Decompress` an, dass Sie dekomprimieren wollen. Am optionalen dritten Argument können Sie bestimmen, dass der Ergebnis-Stream nicht geschlossen wird, wenn Sie den `DeflateStream`- bzw. den `GZipStream` schließen.

Das folgende Beispiel entpackt auf diese Weise eine über den GZIP-Algorithmus komprimierte Datei und schreibt das Ergebnis in eine Zieldatei. Die Quelldatei wird dazu in einen `FileStream` eingelesen. Für die Ergebnisdatei erzeugt das Programm ebenfalls einen `FileStream`. Um den Arbeitsspeicher zu entlasten, werden die Quelldatei blockweise eingelesen und die Daten jeweils an den `GZipStream` übergeben.

Das Beispiel erfordert den Import der Namensräume `System.IO` und `System.IO.Compression`.

```
string sourceFilename = "C:\\Hitchhiker.gz";
string destFilename = "C:\\Hitchhiker.txt";

// Quelldatei in den Quellstream lesen
using (FileStream sourceStream = new FileStream(sourceFilename,
   FileMode.Open))
{
```

Listing 271: Dekomprimieren von Daten mit der GZipStream-Klasse

```
   // Stream für die dekomprimierten Daten erzeugen
   using (FileStream resultStream = new FileStream(destFilename,
      FileMode.Create))
   {
      // GZipStream erzeugen
      using (GZipStream gzipStream = new GZipStream(
         sourceStream, CompressionMode.Decompress))
      {
         // Daten blockweise aus dem Quellstream in den GZipStream lesen
         const int blockSize = 1024 * 1024;
         byte[] buffer = new byte[blockSize];
         int bytesRead;
         while ((bytesRead = gzipStream.Read(buffer, 0, buffer.Length)) != 0)
         {
            resultStream.Write(buffer, 0, bytesRead);
         }
      }
   }
}
```

Listing 271: Dekomprimieren von Daten mit der GZipStream-Klasse (Forts.)

Beim Dekomprimieren von ZIP-Archiven über eine `ZipPackage`-Instanz müssen Sie berücksichtigen, dass diese einzelne logische Dateien beinhalten kann. Über die `PartExists`-Methode können Sie ermitteln, ob eine erwartete logische Datei existiert. Dabei übergeben Sie den URI der Datei (der natürlich identisch zum URI der gespeicherten Datei sein muss). Über `GetPart` können Sie die logische Datei dann auslesen und deren Stream auswerten.

Das folgende Beispiel erfordert die Referenzierung der Assembly *WindowsBase.dll* und den Import der Namensräume `System`, `System.IO` und `System.IO.Packaging`.

```
string sourceFilename = "C:\\Hitchhiker.zip";
string destFilename = "C:\\Hitchhiker.txt";

// URI der erwarteten logischen Datei
Uri packagePartUri = new Uri("/Hitchhiker.txt", UriKind.Relative);

// Quelldatei in den Quellstream lesen
using (FileStream sourceStream = new FileStream(sourceFilename, FileMode.Open))
{
   // Die ZIP-Datei einlesen
   ZipPackage zipPackage = (ZipPackage)ZipPackage.Open(sourceStream,
      FileMode.Open, FileAccess.Read);

   // Überprüfen, ob die erwartete logische Datei existiert
   if (zipPackage.PartExists(packagePartUri))
```

Listing 272: Dekomprimieren von Daten mit der ZipPackage-Klasse

```
    {
        // Die logische Datei auslesen
        ZipPackagePart zipPackagePart = (ZipPackagePart)zipPackage.GetPart(
            packagePartUri);

        // Den Stream auslesen
        Stream packagePartStream = zipPackagePart.GetStream();

        // Stream für die dekomprimierten Daten erzeugen
        using (FileStream resultStream = new FileStream(destFilename,
            FileMode.Create))
        {
            // Daten blockweise aus dem ZIP-Stream in den Ergebnis-Stream lesen
            const int blockSize = 1024 * 1024;
            byte[] buffer = new byte[blockSize];
            int bytesRead;
            while ((bytesRead = packagePartStream.Read(buffer, 0,
                buffer.Length)) > 0)
            {
                resultStream.Write(buffer, 0, bytesRead);
            }
        }
    }
    else
    {
        throw new Exception("Die erwartete logische Datei '" +
            packagePartUri.AbsolutePath + "' existiert nicht im Archiv");
    }
}
```

Listing 272: Dekomprimieren von Daten mit der ZipPackage-Klasse (Forts.)

Hinweis

Alternativ können Sie über die `GetParts`-Methode alle vorhandenen logischen Dateien auslesen und auswerten. Bei der Auswertung helfen der URI der Datei und der MIME-Typ, die Sie aus den Eigenschaften `Uri` und `ContentType` auslesen können. Das überlasse ich aber ganz Ihnen ☺.

163 (ZIP-)Komprimieren von Daten mit #ziplib

Wenn Sie Daten komprimieren und komprimierte Daten wieder entpacken wollen, können Sie, wie ich es bereits in den Rezepten 161 und 162 gezeigt habe, die .NET-Klassen `DeflateStream` und `GZipStream` verwenden. Leider unterstützen diese Klassen nur den Deflate- bzw. den GZIP-Algorithmus, ergeben eine nicht besonders hohe Kompression und bieten keine Unterstützung für die Speicherung von mehreren Dateien in einem Archiv.

Im Internet finden Sie natürlich eine Vielzahl an Bibliotheken, über die Sie Daten komprimieren und komprimierte Daten entpacken können. Eine der besten ist *#ziplib* (*SharpZipLib*). Diese in

C# geschriebene Bibliothek wurde von *Mike Krüger*, dem Entwickler der bekannten Freeware-Entwicklungsumgebung *SharpDevelop*, auf der Basis existierenden Java-Quellcodes programmiert. Sie finden *#ziplib* an der Adresse *www.icsharpcode.net/OpenSource/SharpZipLib*.

#ziplib arbeitet wie `DeflateStream` und `GZipStream` mit einem Stream, weswegen Sie die in dieser Bibliothek enthaltenen Klassen sehr einfach für alle möglichen Daten anwenden können. Im Unterschied zu den beiden genannten .NET-Klassen erlaubt zumindest die Klasse zur Arbeit mit ZIP-Archiven die Einstellung des Kompressionsgrades (von 0 bis 9), das Ablegen von mehreren »Dateien« im erzeugten Archiv, das Erzeugen von Ordnern und das Kommentieren des Archivs.

#ziplib wird unter der GPL-Lizenz (»GNU General Public License«) mit einer Ausnahme veröffentlicht. Kurz gesagt besagt diese Lizenz, dass Entwickler, die unter der GPL-Lizenz entwickeln, immer den Quellcode ihrer Programme mit ausliefern müssen. Andere Programmierer dürfen diesen Quellcode verändern, daraus eigene Programme erzeugen und diese sogar verkaufen. Da dabei immer ein Hinweis auf den Urheber des Programms und die vorgenommenen Veränderungen in das Programm integriert werden müssen, sind die Urheberrechte aller Entwickler geschützt. Ziel dieser Lizenz ist, Betriebssysteme und Programme zu schaffen, deren Quellen frei verfügbar, für jeden zugänglich und nutzbar sind. Auf diese Weise können sich sehr viele Programmierer an der Weiterentwicklung von Programmen und Betriebssystemen beteiligen. Linux, dessen Bestandteile zumeist auch unter der GPL-Lizenz veröffentlicht sind, ist das beste Beispiel für das Funktionieren dieser Idee.

Die Ausnahme von der Standard-Lizenz bei *#ziplib* ist, dass Programme, die diese Bibliothek verwenden, selbst nicht unter der GPL-Lizenz veröffentlicht werden müssen. Dabei wird vorausgesetzt, dass das Programm keine Klassen enthält, die von den *#ziplib*-Klassen abgeleitet werden. So können Sie mit *#ziplib* auch kommerzielle Programme schreiben, die nicht unter die GPL-Lizenz fallen.

#ziplib können Sie über die Assembly *ICSharpCode.SharpZipLib.dll* in ein Programm integrieren, indem Sie diese referenzieren. Die Assembly können Sie über die Adresse *www.icsharpcode.net/OpenSource/SharpZipLib/Download.aspx* herunterladen, indem Sie die Option COMPILED ASSEMBLY WITH DOCUMENTATION AND INSTALL SCRIPTS DOWNLOAD wählen. Im heruntergeladenen Zip-Archiv, das in der zum Zeitpunkt der Erstellung dieses Buch aktuellen Version 0.84 auch auf der Buch-CD gespeichert ist, finden Sie neben der DLL auch eine (leider nicht detaillierte) Windows-Hilfe-Dokumentation. Auf der Website können Sie zusätzlich auch den Quellcode der Bibliothek herunterladen.

#ziplib ist mit einem starken Namen vorbereitet auf die Installation im Globalen Assembly Cache (GAC). Die Datei *installGAC.bat* enthält einen Aufruf des Tools *gacutil*, der diese Installation bewirkt. Beachten Sie, dass die Systemvariable `PATH` den Pfad zum Bin-Ordner des .NET Framework-SDK enthalten muss, damit *gacutil* ohne Pfadangabe aufgerufen werden kann. Falls Sie die Assembly im GAC installieren wollen, sollten Sie diese vor der GAC-Installation in einen Ordner kopieren, den Sie für wiederverwendbare .NET-Klassenbibliotheken vorgesehen haben.

Hinweis

Die Installation im GAC bewirkt, dass mehrere Anwendungen dieselbe Assembly gleichzeitig verwenden können (die Assembly muss dann nicht im Anwendungsordner gespeichert sein). Wenn Sie diese Features nicht benötigen, sollten Sie Assemblys nicht im GAC installieren. Wenn Sie Assemblys im GAC installiert haben, sollten Sie diese laut Microsoft immer über deren originalen Speicherort referenzieren (und nicht über den Assembly-Ordner im Windows-Verzeichnis). Die Installation im GAC bewirkt leider auch nicht, dass die Referenzliste in Visual Studio 2008 diese Assemblys auflistet. Um dies zu erreichen, beachten Sie den folgenden Tipp.

Tipp

Um Assemblys, die nicht im GAC installiert sind, im Referenz-Hinzufügen-Dialog von Visual Studio 2008 verfügbar zu haben, müssen Sie die Registry bearbeiten. Dazu starten Sie das Programm *regedit* im Ausführen-Dialog von Windows oder an der Konsole. Bevor Sie aber an die Arbeit gehen: Beachten Sie, dass Sie mit der Registry sehr vorsichtig umgehen müssen und nur Veränderungen vornehmen sollten, von denen Sie wissen, dass diese keinen Schaden anrichten. Durch fehlerhafte Veränderungen in der Registry können Sie das gesamte System beschädigen (so, jetzt bin ich nicht mehr für Ihr eventuell defektes Windows-System verantwortlich ☺).

Erzeugen Sie einen neuen Schlüssel mit einem eindeutigen Namen für die hinzuzufügenden Assemblys (z. B. *ExternAssemblys*) in der Registry unter dem Schlüssel *HKEY_LOCAL_MACHINE\Software\Microsoft\.NETFramework\AssemblyFolders*. Setzen Sie den Defaultwert dieses Schlüssels auf den Pfad des Ordners, der die Assemblys speichert. Nach einem Neustart von Visual Studio 2008 werden die im Ordner enthaltenen Assemblys in der Liste des Referenz-Hinzufügen-Dialogs aufgeführt.

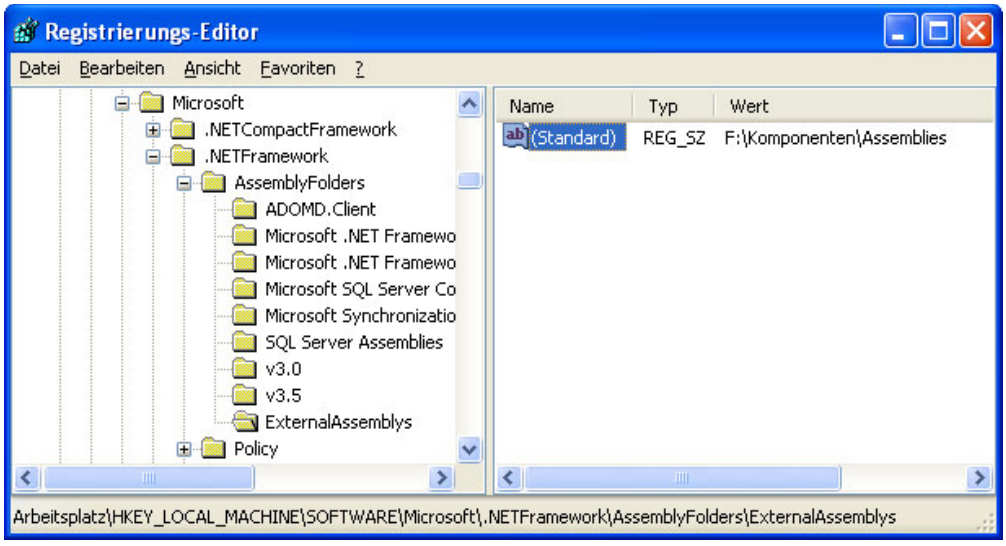


Abbildung 93: Anlegen eines Registry-Schlüssels für das Auflisten der #ziplib-Assembly in der Visual Studio 2008-Liste

Hinweis

Um das Ganze zu vereinfachen, können Sie auch die *.reg*-Datei, die ich der Komponente beigelegt habe, mit einem Texteditor editieren und im Explorer über einen Doppelklick ausführen, um die entsprechenden Schlüssel anzulegen.

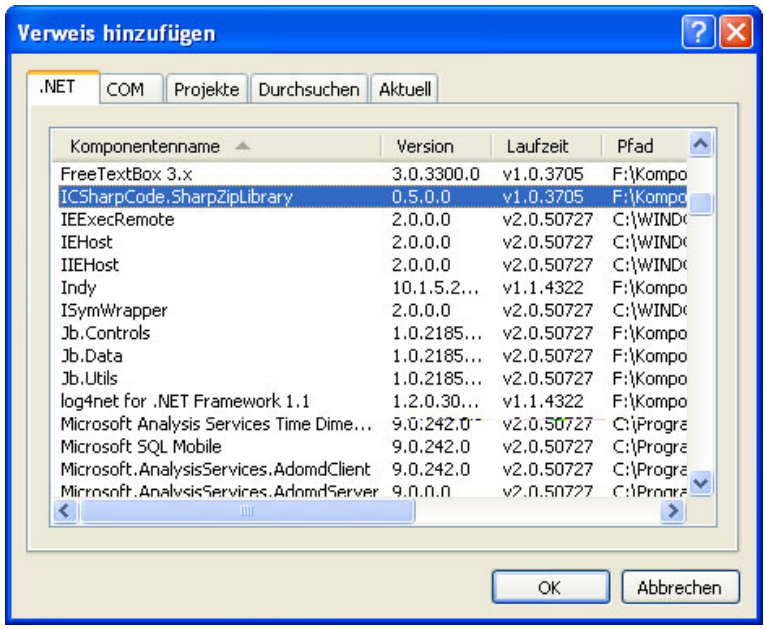


Abbildung 94: Die SharpZipLib-Komponente taucht nach der Registrierung in der Assemblys-Liste des Referenz-Hinzufügen-Dialogs von Visual Studio 2008 auf

#ziplib ist recht komplex und ermöglicht die Arbeit mit *zip*-, *tar*-, *gzip*- und *bzip2*-Archiven. Ich beschränke mich in diesem Buch auf Zip-Archive. Sie können diese Archive im Dateisystem erzeugen, aus Streams »on the fly« erzeugen, im Dateisystem entpacken, die enthaltenen Dateien über Streams lesen und bearbeiten etc. Ich kann hier nicht die vollständige Funktionalität behandeln und zeige deshalb nur die in meinen Augen wichtigsten Features in diesem und den folgenden Rezepten.

Hinweis

Der in diesem Buch vorgestellte Programmcode basiert auf der Version 0.84, die der Buch-CD beiliegt. Wenn Sie die Komponente aus dem Internet herunterladen, handelt es sich wahrscheinlich um eine neuere Version, die u. U. mit der Version 0.84 nicht mehr kompatibel ist. Es kann also sein, dass Sie den folgenden Programmcode mit der aktuellen Version nur unter einigen Anpassungen verwenden können.

Um ein Zip-Archiv zu erzeugen, erzeugen Sie zunächst eine Instanz der Klasse `ICSharpCode.SharpZipLib.Zip.ZipOutputStream`. Im Konstruktor übergeben Sie ein `Stream`-Objekt, in das die komprimierten Daten geschrieben werden sollen. Hier können Sie natürlich (wie bei den Klassen `DeflateStream` und `GZipStream` aus dem Rezept 161) jeden beliebigen Stream übergeben. Dieser darf lediglich nicht schreibgeschützt sein. So können Sie das Ergebnis zum Bei-

spiel über einen FileStream in einer Datei schreiben oder in einer Webanwendung in den Response-Stream.

Über die Methode SetLevel können Sie die Kompressionsrate mit Werten zwischen 0 und 9 definieren. Eine Kompressionsrate von 0 bewirkt keine Komprimierung, 9 führt zu einer maximalen Komprimierung (was aber auch den Archiviervorgang verlangsamt). Die (nicht dokumentierte) Voreinstellung ist 6.

Über die Methode SetComment können Sie einen Kommentar im Archiv speichern.

Dann erzeugen Sie für jede hinzuzufügende Datei ein ZipEntry-Objekt, das einen Datei- oder Ordnereintrag in der Zip-Datei repräsentiert. Dieses Objekt initialisieren Sie im Konstruktor mit dem Dateinamen, so wie dieser im Archiv gespeichert werden soll. Hier können Sie auch (relative) Pfadangaben verwenden, um Dateien so zu archivieren, dass diese beim Entpacken in Unterordnern gespeichert werden. In der Eigenschaft Comment können Sie einen Kommentar zu der archivierten Datei ablegen. Dieses Objekt fügen Sie dem ZipOutputStream-Objekt dann über die Methode PutNextEntry hinzu.

Nun müssen Sie noch die eigentlichen Daten in den Stream schreiben, was wie bei jedem anderen Stream auch über die Write-Methode geschieht.

Das Erzeugen des ZipEntry-Objekts und das Hinzufügen der Datei in den ZipOutputStream wiederholen Sie für jede hinzuzufügende Datei. Schließlich rufen Sie noch die Finish-Methode des ZipEntry-Objekts auf um die Zip-Datei zu erzeugen, und schließen den ZipOutputStream.

Das folgende Beispiel liest zwei Dateien in je ein FileStream-Objekt ein und archiviert diese in einen ZipOutputStream, der als Basis-Stream ebenfalls einen FileStream verwendet. Das Beispiel erzeugt damit eine Zip-Datei.

Zum Kompilieren müssen Sie die Assembly *ICSharpCode.SharpZipLib.dll* referenzieren und die Namensräume System, System.IO, System.Reflection und ICSharpCode.SharpZipLib.Zip importieren.

```
// Erste Quelldatei in einen Quellstream lesen
string applicationPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string sourceFilename = Path.Combine(applicationPath, "Hitchhiker.txt");
FileStream sourceStream1 = new FileStream(sourceFilename, FileMode.Open);

// Zweite Quelldatei in einen weiteren Quellstream lesen
sourceFilename = Path.Combine(applicationPath, "killingme.mp3");
FileStream sourceStream2 = new FileStream(sourceFilename, FileMode.Open);

// Stream für die komprimierten Daten erzeugen
string destFilename = Path.Combine(applicationPath, "Demo.zip");
FileStream resultStream = new FileStream(destFilename, FileMode.Create);
```

Listing 273: Komprimieren von Daten mit #ZipLib

```
// ZipOutputStream erzeugen
ZipOutputStream zipStream = new ZipOutputStream(
   resultStream);

// Kompressions-Level auf maximale Kompression setzen
zipStream.SetLevel(9);

// ZipEntry-Objekt für die erste Datei hinzufügen
zipStream.PutNextEntry(new ZipEntry("Hitchhiker.txt"));

// Daten der ersten Datei blockweise aus dem
// Quellstream in den ZipOutputStream lesen
const int blockSize = 1024;
byte[] buffer = new byte[blockSize];
int bytesRead;
while ((bytesRead = sourceStream1.Read(buffer, 0, buffer.Length)) > 0)
{
   zipStream.Write(buffer, 0, bytesRead);
}

// ZipEntry-Objekt für die zweite Datei hinzufügen
zipStream.PutNextEntry(new ZipEntry("killingme.mp3"));

// Daten der zweiten Datei blockweise aus dem
// Quellstream in den ZipOutputStream lesen
buffer = new byte[blockSize];
while ((bytesRead = sourceStream2.Read(buffer, 0, buffer.Length)) > 0)
{
   zipStream.Write(buffer, 0, bytesRead);
}

// ZipOutputStream abschließen
zipStream.Finish();

// Quellstreams schließen
sourceStream1.Close();
sourceStream2.Close();

// ZipOutputStream (und damit automatisch auch den
// Ergebnis-Stream) schließen
zipStream.Close();
```

Listing 273: Komprimieren von Daten mit #ZipLib (Forts.)

164 (ZIP-)Dekomprimieren von Daten mit #ziplib

Zum Dekomprimieren von Daten, die das ZIP-Format verwenden, können Sie die Bibliothek *#ziplib* einsetzen. Die Grundlagen zur Verwendung dieser Bibliothek habe ich bereits im Rezept 163 behandelt.

Die zu dekomprimierenden Daten müssen in einem Stream vorliegen. Diesen übergeben Sie einer neuen Instanz der Klasse `ZipInputStream`. Da die einzelnen Dateien in ZIP-Archiven über einzelne `ZipEntry`-Objekte repräsentiert werden, gehen Sie über die `GetNextEntry`-Methode alle im Archiv enthaltenen `ZipEntry`-Objekte durch. Über diese Objekte ermitteln Sie den Dateinamen der Datei und gegebenenfalls einen angegebenen Unterordner.

Da der `ZipInputStream` nach dem Lesen eines `ZipEntry`-Objekts auf dem Anfang der Datei steht, können Sie diese danach über die `Read`-Methode des Streams in ein Byte-Array lesen. Dabei brauchen Sie sich natürlich nicht um das Dekomprimieren der gepackten Daten zu kümmern.

Achtung

Ein `ZipEntry`-Objekt liefert über die Eigenschaft `CompressedSize` eine Information über die Größe der komprimierten Daten. Die Eigenschaft `Size` liefert die Größe der unkomprimierten Daten. Theoretisch könnten Sie die über ein `ZipEntry`-Objekt repräsentierten Daten in einem Rutsch in einen Byte-Array einlesen. Dieses Array müsste dann die Größe besitzen, die `Size` angibt. Leider ist dies in der Praxis scheinbar nicht möglich: In meinen Tests mit zwei Dateien in einem ZIP-Archiv wurde die erste nicht eingelesen, beim Eintritt in den zweiten Durchlauf der Schleife durch die `ZipEntry`-Objekte generierte die `GetNextEntry`-Methode sogar die Ausnahme »Wrong local header signature«.

Um dieses Problem zu vermeiden, sollten Sie die Daten immer in einer Schleife blockweise einlesen (was bei großen Dateien auch den Arbeitsspeicherverbrauch reduziert). `Read` gibt die Anzahl der gelesenen Bytes zurück. Die Schleife muss also so lange ausgeführt werden, bis `Read` 0 zurückgibt. Die einzelnen Byte-Blöcke können Sie dann zum Beispiel in ein `FileStream`-Objekt schreiben um eine Datei zu erzeugen.

Das Beispiel in Listing 274 liest zuerst eine ZIP-Datei in ein `FileStream`-Objekt ein. Dieser Stream dient als Basis für das danach erzeugte `ZipInputStream`-Objekt. In einer Schleife werden dann alle `ZipEntry`-Objekte ermittelt und über diese der Name der jeweils im Archiv gespeicherten Datei. Das Programm erzeugt für jede Datei ein `FileStream`-Objekt, in das die blockweise eingelesenen Daten geschrieben werden.

Zum Kompilieren dieses Beispiels müssen Sie die Assembly *ICSharpCode.SharpZipLib.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Reflection` und `ICSharpCode.SharpZipLib.Zip` importieren.

```
// Ordner für die Dateien ermitteln
string sourcePath =
   Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
string destPath = sourcePath;
```

Listing 274: Entpacken von ZIP-komprimierten Daten

```
// Dateiname der ZIP-Datei ermitteln
string zipFileName = Path.Combine(sourcePath, "Demo.zip");

// ZIP-Datei in einen FileStream lesen
FileStream sourceStream = new FileStream(zipFileName, FileMode.Open);

// ZipInputStream für die Zip-Datei erzeugen
ZipInputStream zipInputStream = new ZipInputStream(sourceStream);

  // Alle im Archiv gespeicherten ZipEntry-Objekte durchgehen
ZipEntry zipEntry;
while ((zipEntry = zipInputStream.GetNextEntry()) != null)
{
   // Dateinamen (inklusive eventuellem relativen Pfad) auslesen
   // Für das Beispiel wird nur der Dateiname verwertet
   string resultFilename = Path.GetFileName(zipEntry.Name);

   // FileStream für die Datei erzeugen
   FileStream resultStream =
      File.Create(Path.Combine(destPath, resultFilename));

   // Datei in Blöcken von maximal 1 MB in den Stream schreiben
   // um den Speicher nicht mit großen Dateien zu überlasten
   int bytesRead;
   byte[] buffer = new byte[1048576];
   do
   {
      // Den nächsten Datenblock aus dem ZipInputStream lesen
      bytesRead = zipInputStream.Read(buffer, 0, buffer.Length);
      if (bytesRead > 0)
      {
         // Wenn Daten gelesen wurden, diese in den
         // Ergebnis-Stream schreiben
         resultStream.Write(buffer, 0, bytesRead);
      }
   } while (bytesRead > 0);

   // Ergebnis-Stream schließen
   resultStream.Close();
}

// ZipInputStream schließen
zipInputStream.Close();
```

Listing 274: Entpacken von ZIP-komprimierten Daten (Forts.)

165 Dateien in ZIP-Archive komprimieren

Wie ich es in Rezept 163 gezeigt habe, ist das Erzeugen eines ZIP-Archivs sehr einfach. In diesem Rezept entwickle ich nun eine Methode, die eine beliebige Anzahl von Dateien in eine ZIP-Datei komprimiert. Das Prinzip dieser Methode ist dasselbe wie im genannten Rezept.

Die Methode `ZipFiles` in Listing 275 schreibt mehrere in einem String-Array übergebene Dateien in ein Archiv. Übergeben wird zunächst das Array mit den Quelldateinamen, der Name der zu erzeugenden Zip-Datei und die Blockgröße in Byte.

Das Argument `zipLevel` legt die Kompressionsrate fest. Sie können hier einen Wert von 0 (keine Kompression) bis 9 (maximale Kompression) eintragen. Eine höhere Kompression verlangsamt natürlich den Archiviervorgang.

Am Argument `includePaths` wird festgelegt, ob der Pfad der Datei mit gespeichert werden soll. In Zip-Archiven können Dateien ohne, mit einer relativen oder einer absoluten Pfadangabe archiviert werden. Die Angabe von Pfaden bringt beim normalen Entpacken eines Archivs den Vorteil, dass die Dateien wieder in einen entsprechenden Ordner entpackt werden. Um möglichst flexibel zu sein, können Sie beim Aufruf von `ZipFiles` entscheiden, ob der Pfad jeder Datei mit aufgenommen werden soll. Soll der Pfad mit aufgenommen werden, entfernt `ZipFiles` aber den Laufwerkbuchstaben aus der Pfadangabe, da auf dem Zielsystem nicht vorausgesetzt werden kann, dass das entsprechende Laufwerk existiert.

Am letzten Argument können Sie einen Kommentar zur Zip-Datei übergeben, den die meisten Zip-Programme beim Lesen eines Archivs automatisch anzeigen.

Zum Kompilieren dieser Methode müssen Sie die Assembly *ICSharpCode.SharpZipLib.dll* referenzieren und die Namensräume `System.IO` und `ICSharpCode.SharpZipLib.Zip` importieren.

```
public static void ZipFiles(string[] sourceFileNames, string zipFileName,
   int blockSize, int zipLevel, bool includePaths, string comment)
{
   // Datei-Stream als Basis-Stream erzeugen
   Stream zipFileStream = File.Open(zipFileName, FileMode.CreateNew);

   // ZipOutputStream zum Schreiben der Zip-Datei erzeugen
   ZipOutputStream zipOutputStream = new ZipOutputStream(zipFileStream);

   // Kompressionsrate definieren (0 bis 9)
   zipOutputStream.SetLevel(zipLevel);

   // Kommentar zum Archiv definieren
   if (comment != null)
   {
      zipOutputStream.SetComment(comment);
   }
```

Listing 275: Methode zum Erzeugen eines Zip-Archivs über #ziplib

```
// Alle im Array sourceFileNames übergebenen Dateien
// durchgehen und in das Archiv schreiben
for (int i = 0; i < sourceFileNames.Length; i++)
{
   // ZipEntry-Objekt für die neue Datei erzeugen. Der im Konstruktor
   // übergebene Name wird als Dateiname beim Extrahieren verwendet.
   // Sie können hier auch (relative) Pfadangaben mit angeben.
   // Das Programm speichert den Pfad (ohne Laufwerkangabe) mit,
   // wenn das Argument includePaths true ist
   string fileNameForZip;
   FileInfo fi = new FileInfo(sourceFileNames[i]);
   if (includePaths)
   {
      // Dateiname ohne Laufwerkbuchstabe ermitteln
      int pos = fi.FullName.IndexOf('\\');
      if (pos > -1)
      {
         fileNameForZip = fi.FullName.Substring(pos,
            fi.FullName.Length - pos);
      }
      else
      {
         fileNameForZip = fi.FullName;
      }
   }
   else
   {
      // Nur den Dateinamen speichern
      fileNameForZip = fi.Name;
   }
   ZipEntry zipEntry = new ZipEntry(fileNameForZip);

   // ZipEntry-Objekt dem ZipOutputStream hinzufügen
   zipOutputStream.PutNextEntry(zipEntry);

   // Zu archivierende Datei in einem FileStream öffnen
   FileStream fileStream = new FileStream(sourceFileNames[i],
      FileMode.Open, FileAccess.Read);

   // Quellstream blockweise in ein Byte-Array lesen und in den
   // ZipOutputStream schreiben
   byte[] buffer = new byte[blockSize];
   int bytesRead = 0;
   while ((bytesRead = fileStream.Read(buffer, 0, buffer.Length)) > 0)
   {
```

Listing 275: Methode zum Erzeugen eines Zip-Archivs über #ziplib (Forts.)

```
            zipOutputStream.Write(buffer, 0, bytesRead);
        }

        // FileStream schließen
        fileStream.Close();
    }

    // ZipOutputStream abschließen und schließen
    zipOutputStream.Finish();
    zipOutputStream.Close();
}
```

Listing 275: Methode zum Erzeugen eines Zip-Archivs über #ziplib (Forts.)

Das folgende Beispiel zeigt die Anwendung dieser Methode. Ausnahmen, die beim Erzeugen des Archivs auftreten, werden dabei abgefangen. Das Beispiel benötigt neben den Voraussetzungen der `ZipFiles`-Methode den Import der Namensräume `System`, `System.IO` und `System.Reflection`.

```
// Dateiname der Zip-Datei
string zipFileName = "c:\\Hitchhiker.zip";

// Zip-Datei löschen, falls diese bereits vorhanden ist
if (File.Exists(zipFileName))
{
    File.Delete(zipFileName);
}

// Dateien, die archiviert werden sollen
string applicationPath = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
string[] sourceFiles = {
    Path.Combine(applicationPath, "Hitchhiker.txt"),
    Path.Combine(applicationPath, "Hitchhiker.jpg")};

try
{
    // Dateien mit einer Blockgröße von 1 MB und einer
    // Kompressionsrate von 9 archivieren.
    // Pfade sollen nicht mit eingeschlossen werden
    ZipFiles(sourceFiles, zipFileName, 1048576,
        9, false, "Demo für die Verwendung von #ziplib");

    Console.WriteLine("Fertig");
}
catch (Exception ex)
```

Listing 276: Anwendung der Methode ZipFiles zum Archivieren von Dateien

```
{
   Console.WriteLine("Fehler beim Archivieren: {0}", ex.Message);
}
```

Listing 276: Anwendung der Methode ZipFiles zum Archivieren von Dateien (Forts.)

166 (ZIP-)Archive aus einem Ordner erzeugen

Über *#ziplib* (siehe Rezept 163) können Sie recht einfach ganze Ordner archivieren. Dazu müssen Sie lediglich den Ordner rekursiv durchgehen und die Dateien aller enthaltenen Unterordner separat archivieren. Um die Ordnerstruktur beim Entpacken beizubehalten, sollten Sie die Datei-Einträge im Archiv mit relativen Pfadangaben speichern. Die im Folgenden beschriebene Methode `ZipFolder` ruft dazu die rekursive Methode `AddFilesFromFolder` auf, die die Dateien eines Ordners archiviert und sich dann selbst für alle Unterordner aufruft.

`ZipFolder` erwartet als Argumente zunächst den Ordnernamen und den Namen der zu erzeugenden Archivdatei. Am dritten Argument übergeben Sie wie im vorhergehenden Rezept eine Blockgröße, die `ZipFolder` zum blockweisen Lesen und Schreiben der Streams verwendet. Mit einer nicht zu großen Blockgröße verhindern Sie Arbeitsspeicherprobleme bei sehr großen Dateien.

Das Argument `zipLevel` definiert die Kompressionsrate mit einem Wert zwischen 0 (keine Kompression) und 9 (maximale Kompression).

Am Argument `comment` können Sie einen Kommentar für die Zip-Datei übergeben.

Das letzte (params-)Argument dient der Angabe der Endungen von Dateien, die in das Archiv einzuschließen sind. Ist dieses Argument nicht angegeben, werden grundsätzlich alle Dateien aus dem angegebenen Ordner in das Archiv eingeschlossen. Für den Fall, dass Sie nur bestimmte Dateien in das Archiv übernehmen wollen, geben Sie hier die Endungen (inklusive Punkt) an. Groß- und Kleinschreibung wird dabei nicht unterschieden.

`ZipFolder` erfordert die Referenzierung der Assembly *ICSharpCode.SharpZipLib.dll* und den Import der Namensräume `System`, `System.IO` und `ICSharpCode.SharpZipLib.Zip`.

```
/* Archiviert einen Ordner in eine ZIP-Datei */
public static void ZipFolder(string folderName, string zipFileName,
   int blockSize, int zipLevel, string comment,
   params string[] extensionsToInclude)
{
   // Datei-Stream als Basis-Stream erzeugen
   Stream zipFileStream = File.Open(zipFileName, FileMode.CreateNew);

   // ZipOutputStream zum Schreiben der Zip-Datei erzeugen
   ZipOutputStream zipOutputStream = new ZipOutputStream(zipFileStream);
```

Listing 277: Methode zum rekursiven Archivieren eines Ordners über #ziplib

```
   // Kompressionsrate definieren (0 bis 9
   zipOutputStream.SetLevel(zipLevel);

   // Kommentar zum Archiv definieren
   if (comment != null)
   {
      zipOutputStream.SetComment(comment);
   }

   // Ordner rekursiv archivieren
   AddFilesFromFolder(folderName, folderName, zipOutputStream, blockSize,
      extensionsToInclude);

   // ZipOutputStream abschließen und schließen
   zipOutputStream.Finish();
   zipOutputStream.Close();
}

/* Fügt alle Dateien eines Ordners (rekursiv) einem ZIP-Archiv hinzu */
private static void AddFilesFromFolder(string baseFolderName,
   string folderName, ZipOutputStream zipOutputStream,
   int blockSize, params string[] extensionsToInclude)
{
   // Alle Dateien des Ordners durchgehen und in das Archiv schreiben
   DirectoryInfo folder = new DirectoryInfo(folderName);
   FileInfo[] files = folder.GetFiles();
   for (int i = 0; i < files.Length; i++)
   {
      // Überprüfen, ob Erweiterungen für einzuschließende Dateien
      // angegeben sind
      bool includeFile = true;
      if (extensionsToInclude != null && extensionsToInclude.Length > 0)
      {
         includeFile = false;
         foreach (string extensionToInclude in extensionsToInclude)
         {
            if (files[i].Extension.ToLower() == extensionToInclude.ToLower())
            {
               includeFile = true;
               break;
            }
         }
      }

      if (includeFile == false)
      {
```

Listing 277: Methode zum rekursiven Archivieren eines Ordners über #ziplib (Forts.)

```
            continue;
         }

         // Relativen Pfad des aktuellen Ordners über das Entfernen des
         // Basisordnernamens ermitteln
         string relativePath = folderName.Replace(baseFolderName, "");
         if (relativePath != null)
         {
            if (relativePath.StartsWith("\\"))
               relativePath = relativePath.Remove(0, 1);
            if (relativePath.EndsWith("\\") == false)
               relativePath += "\\";
         }

         // ZipEntry-Objekt für die neue Datei mit dem relativen Pfad
         // erzeugen und dem ZipOutputStream hinzufügen
         zipOutputStream.PutNextEntry(
            new ICSharpCode.SharpZipLib.Zip.ZipEntry(
            relativePath + files[i].Name));

         // Zu archivierende Datei in einem FileStream öffnen und über ein
         // Byte-Array blockweise in den ZipOutputStream schreiben
         FileStream fileStream = files[i].OpenRead();
         byte[] buffer = new byte[blockSize];
         int bytesWritten = 0;
         do
         {
            int size = fileStream.Read(buffer, 0, buffer.Length);
            zipOutputStream.Write(buffer, 0, size);
            bytesWritten += size;

         } while (bytesWritten < fileStream.Length);

         // FileStream schließen
         fileStream.Close();
      }

      // Alle Unterordner durchgehen und die Methode rekursiv aufrufen
      DirectoryInfo[] subFolders = folder.GetDirectories();
      for (int i = 0; i < subFolders.Length; i++)
      {
         AddFilesFromFolder(baseFolderName, subFolders[i].FullName,
            zipOutputStream, blockSize);
      }
   }
```

Listing 277: Methode zum rekursiven Archivieren eines Ordners über #ziplib (Forts.)

Das folgende Beispiel zeigt eine Anwendung der `ZipFolder`-Methode.

```
ZipFolder("c:\\inetpub", "c:\\demo.zip", 1048576, 9, "");
```

Das rekursive Archivieren eines Ordners über *ZipFolder* funktioniert sehr gut (Abbildung 95), ist allerdings wesentlich langsamer als das Archivieren mit einem professionellen Archivier-Programm wie WinZip ☺. Vielleicht können Sie ja noch ein wenig optimieren ...

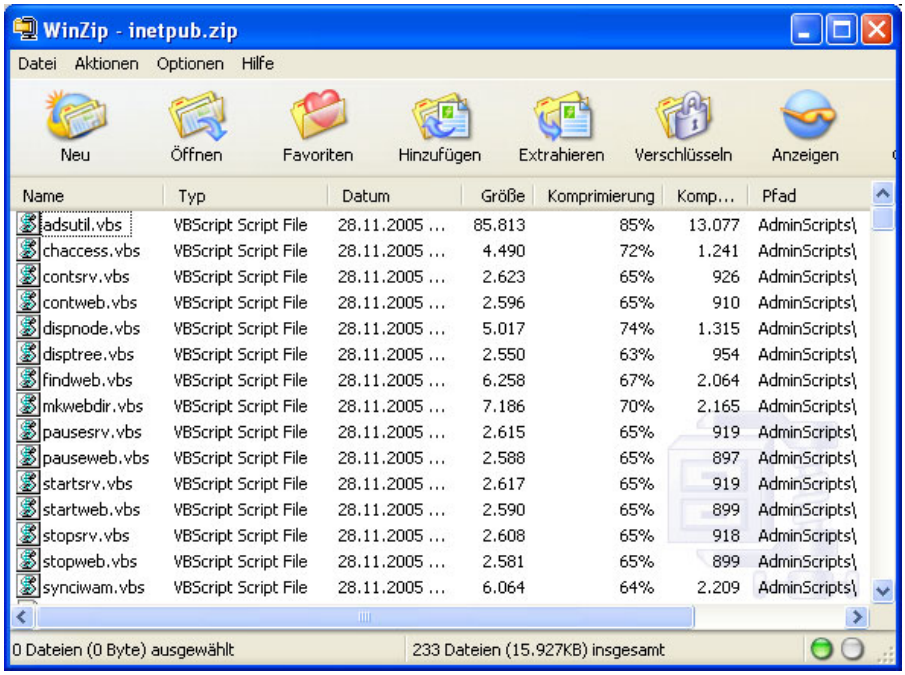


Abbildung 95: Ein mit #ziplib rekursiv archivierter Ordner in WinZip

167 Dateiinformationen aus (ZIP-)Archiven auslesen

Um Informationen über die in einem Zip-Archiv enthaltenen Dateien auszulesen, können Sie (natürlich ☺) die Komponente *#ziplib* verwenden (siehe Rezept 163). Dazu erzeugen Sie zunächst eine `ZipInputStream`-Instanz zum Einlesen der Archivdatei. Wie schon beim Erzeugen eines Archivs, übergeben Sie im Konstruktor einen normalen Stream, der die zu lesende Datei repräsentiert. Dann können Sie über die Methode `GetNextEntry` in einer Schleife das jeweils nächste `ZipEntry`-Objekt auslesen. `ZipEntry`-Objekte speichern die Informationen zu einer im Archiv gespeicherten Datei oder einem Ordner. Über die Eigenschaft `Name` können Sie den Namen der Datei bzw. des Ordners auslesen, die Eigenschaft `Size` verwaltet die unkomprimierte Größe der Datei, aus der Eigenschaft `CompressedSize` können Sie die komprimierte Größe auslesen. Die Eigenschaft `IsDirectory` gibt Auskunft darüber, ob es sich bei dem Eintrag um einen Ordner handelt. `ZipEntry`-Objekte besitzen noch weitere Eigenschaften, von denen ich die wichtigsten in Listing 278 verwende.

Das Beispiel erfordert die Referenzierung der Assembly *ICSharpCode.SharpZipLib.dll* und den Import der Namensräume `System`, `System.IO`, `System.Reflection` und `ICSharpCode.SharpZipLib.Zip`.

```
// Pfad und Dateiname ermitteln
string applicationPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string zipFileName = Path.Combine(applicationPath, "Demo.zip");

// ZipInputStream für die Zip-Datei erzeugen
ZipInputStream zipInputStream = new ZipInputStream(
   File.Open(zipFileName, FileMode.Open, FileAccess.Read));

// Alle im Archiv gespeicherten ZipEntry-Objekte durchgehen
ZipEntry zipEntry;
while ((zipEntry = zipInputStream.GetNextEntry()) != null)
{
   Console.WriteLine("Name : {0}", zipEntry.Name);
   Console.WriteLine("Größe : {0}", zipEntry.Size);
   Console.WriteLine("Komprimierte Größe : {0}", zipEntry.CompressedSize);
   Console.WriteLine("Kompressionsmethode : {0}",
      zipEntry.CompressionMethod);
   Console.WriteLine("Dateidatum : {0}", zipEntry.DateTime.ToString());
   Console.WriteLine("Kommentar : {0}", zipEntry.Comment);
   Console.WriteLine("CRC : {0}", zipEntry.Crc);
   Console.WriteLine();
}

// ZipInputStream schließen
zipInputStream.Close();
```

Listing 278: Auslesen von Informationen über die in einem Zip-Archiv gespeicherten Dateien

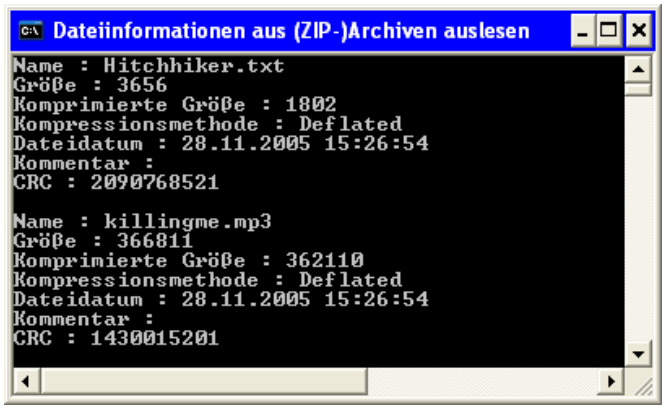


Abbildung 96: Das Beispielprogramm zum Auslesen von Informationen über die in einem Zip-Archiv gespeicherten Dateien

168 (ZIP-)Archive entpacken

Das Entpacken von Zip-Archiven ist über *#ziplib* (siehe Rezept 163) ebenso einfach wie das Verpacken. Wie im vorhergehenden Rezept erzeugen Sie dazu eine Instanz der Klasse `ZipInputStream` und gehen über die `GetNextEntry`-Methode alle `ZipEntry`-Objekte durch. Über diese Objekte ermitteln Sie den Dateinamen der jeweiligen im Archiv gespeicherten Datei und den eventuell angegebenen Unterordner. Ist ein Ordner angegeben, sollten Sie diesen zunächst im Zielordner anlegen. Da der `ZipInputStream` nach dem Lesen eines `ZipEntry`-Objekts auf dem Anfang der Datei steht, können Sie diese danach über die `Read`-Methode des Streams in ein Byte-Array lesen. Dabei brauchen Sie sich natürlich nicht um das Dekomprimieren der gepackten Daten zu kümmern.

Achtung

Ein `ZipEntry`-Objekt liefert über die Eigenschaft `CompressedSize` eine Information über die Größe der komprimierten Daten. Die Eigenschaft `Size` liefert die Größe der unkomprimierten Daten. Theoretisch könnten Sie die über ein `ZipEntry`-Objekt repräsentierten Daten in einem Rutsch in einen Byte-Array einlesen. Dieses Array müsste dann die Größe besitzen, die `Size` angibt. Leider ist dies in der Praxis scheinbar nicht möglich: In meinen Tests mit zwei Dateien in einem ZIP-Archiv wurde die erste nicht eingelesen, beim Eintritt in den zweiten Durchlauf der Schleife durch die `ZipEntry`-Objekte generierte die `GetNextEntry`-Methode sogar die Ausnahme »Wrong local header signature«.

Um dieses Problem zu vermeiden, sollten Sie die Daten immer in einer Schleife blockweise einlesen. `Read` gibt die Anzahl der gelesenen Bytes zurück. Die Schleife muss also so lange ausgeführt werden, bis `Read` 0 zurückgibt. Die einzelnen Byte-Blöcke können Sie dann zum Beispiel in ein `FileStream`-Objekt schreiben um eine Datei zu erzeugen.

Das Lesen in einer Schleife besitzt noch einen anderen Vorteil. Bei sehr großen Dateien können Sie diese so in kleineren Teilen einlesen, um den Arbeitsspeicher nicht zu überlasten. Die einzelnen Byte-Blöcke können Sie dann in ein `FileStream`-Objekt schreiben um eine Datei zu erzeugen.

Die Methode `ExtractToFolder` in Listing 279 extrahiert alle Dateien eines Archivs in einen Ordner. Die Dateien werden dabei in Blöcken einer festzulegenden Größe aus dem Archiv gelesen. Um bereits vorhandene Dateien nicht ohne Warnung zu überschreiben, fragt `ExtractToFolder` in diesem Fall beim Anwender nach, ob er diese überschreiben will. Antwortet der Anwender mit Ja, fragt die Methode einmalig nach, ob alle Dateien überschrieben werden sollen. Diese Überschreib-Logik stammt aus dem Rezept 136. Die Nachfrage kann über das Argument `overwriteWithoutWarning` auch abgeschaltet werden.

`ExtractToFolder` berücksichtigt die Tatsache, dass Dateien in einigen ZIP-Archiven auch mit einem Backslash als Ordner bezeichnet sein können. In diesem Fall entfernt die Methode den Unterordner einfach (da ein einzelner Backslash einen Wurzel-Ordner bezeichnet). Außerdem berücksichtigt `ExtractToFolder` die Tatsache, dass ein Archiv auch 0-Byte-Dateien beinhalten kann. Diese werden dann natürlich nicht aus dem Archiv gelesen, aber im Ziel-Ordner erzeugt.

`ExtractToFolder` erfordert die Referenzierung der Assemblys *ICSharpCode.SharpZipLib.dll* und *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Windows.Forms` und `ICSharpCode.SharpZipLib.Zip`.

```
/* Merker für das Überschreiben aller Dateien */
private static bool overwriteAllFiles;
private static bool alreadyAskedForOverwriteAllFiles;

/* Extrahiert die Dateien eines ZIP-Archivs in einen Ordner */
public static void ExtractToFolder(string zipFileName, string folderName,
   int blockSize, bool overwriteWithoutWarning)
{
   // Felder voreinstellen
   overwriteAllFiles = false;
   alreadyAskedForOverwriteAllFiles = false;

   // ZipInputStream für die Zip-Datei erzeugen
   ZipInputStream zipInputStream = new ZipInputStream(
      File.Open(zipFileName, FileMode.Open, FileAccess.Read));

   // Alle im Archiv gespeicherten ZipEntry-Objekte durchgehen
   ZipEntry zipEntry;
   while ((zipEntry = zipInputStream.GetNextEntry()) != null)
   {
      // Aus dem (relativen) Dateinamen den Unterordner und den
      // Namen extrahieren
      string subFolderName = Path.GetDirectoryName(zipEntry.Name);

      // Für den Fall, dass die Zip-Einträge mit einem \ beginnen,
      // den Unterordner auf null setzen
      if (subFolderName == "\\")
      {
         subFolderName = null;
      }

      // Den Dateinamen des Eintrags ermitteln
      string entryName = Path.GetFileName(zipEntry.Name);

      // Den vollen Ordnernamen des Zielordners ermitteln
      string destFolderName;
      if (subFolderName != null)
      {
         destFolderName = Path.Combine(folderName, subFolderName);
      }
      else
      {
         destFolderName = folderName;
      }
```

Listing 279: Methode zum Extrahieren eines Zip-Archivs über #ziplib

```
// Unterordner erzeugen, falls notwendig
if (subFolderName != null)
{
   Directory.CreateDirectory(destFolderName);
}

if (zipEntry.IsDirectory == false &&
   entryName != null && entryName.Length > 0)
{
   bool overwriteFile = true;

   if (overwriteWithoutWarning == false &&
      overwriteAllFiles == false)
   {
      // Wenn Dateien nicht ohne Warnung überschrieben werden
      // sollen: Überprüfen, ob bereits eine gleichnamige Datei
      // existiert
      if (File.Exists(Path.Combine(destFolderName, entryName)))
      {
         // Nachfragen, ob die Datei überschrieben werden soll
         switch (MessageBox.Show("Die Datei '" + entryName +
            "' existiert bereits im Ordner '" + destFolderName +
            "'\r\n\r\nSoll diese Datei überschrieben werden?",
            Application.ProductName, MessageBoxButtons.YesNoCancel,
            MessageBoxIcon.Question))
         {
            case DialogResult.Yes:
               overwriteFile = true;
               break;

            case DialogResult.No:
               overwriteFile = false;
               break;

            case DialogResult.Cancel:
               // Stream schließen und beenden
               zipInputStream.Close();
               return;
         }

         // Nachfragen, ob alle Dateien überschrieben werden
         // sollen, sofern dies noch nicht geschehen ist
         if (overwriteFile == true &&
            alreadyAskedForOverwriteAllFiles == false)
         {
```

Listing 279: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

```
            switch (MessageBox.Show("Sollen alle vorhandenen " +
               "Dateien automatisch überschrieben werden?",
               Application.ProductName, MessageBoxButtons.YesNoCancel,
               MessageBoxIcon.Question))
            {
               case DialogResult.Yes:
                  overwriteAllFiles = true;
                  break;

               case DialogResult.No:
                  overwriteAllFiles = false;
                  break;

               case DialogResult.Cancel:
                  // Stream schließen und beenden
                  zipInputStream.Close();
                  return;
            }
            // Definieren, dass nicht noch einmal gefragt wird
            alreadyAskedForOverwriteAllFiles = true;
         }
      }
   }

   if (overwriteFile)
   {
      // FileStream für die Datei erzeugen
      FileStream fileStream =
         File.Create(Path.Combine(destFolderName, entryName));

      if (zipEntry.CompressedSize > 0)
      {
         // Datei in Blöcken von maximal 1 MB in den Stream schreiben
         // um den Speicher nicht mit großen Dateien zu überlasten
         int size;
         byte[] buffer = new byte[1048576];
         do
         {
            // Den nächsten Datenblock aus dem ZipInputStream lesen
            size = zipInputStream.Read(buffer, 0, buffer.Length);
            if (size > 0)
            {
               // Wenn Daten gelesen wurden, diese in die Datei
               // schreiben
               fileStream.Write(buffer, 0, size);
```

Listing 279: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

```
                }
            } while (size > 0);
        }

        // FileStream schließen
        fileStream.Close();
    }
  }
}

// ZipInputStream schließen
zipInputStream.Close();
}
```

Listing 279: Methode zum Extrahieren eines Zip-Archivs über #ziplib (Forts.)

Die Anwendung dieser Methode ist wieder einmal sehr einfach:

```
// Pfad und Dateiname ermitteln
string sourcePath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string zipFileName = Path.Combine(sourcePath, "Demo.zip");
string destPath = "C:\\Temp\\Demo";

// Zip-Archiv mit einer Blockgröße von 1 MB
// mit Überschreib-Nachfrage entpacken
Console.WriteLine("Extrahiere {0} nach {1} ... ", zipFileName, destPath);
try
{
   ExtractToFolder(zipFileName, destPath, 1048576, false);
}
catch (ZipException ex)
{
   Console.WriteLine("Fehler bei der Archivoperation: {0}", ex.Message);
}
catch (IOException ex)
{
   Console.WriteLine("Fehler bei der Dateioperation: {0}", ex.Message);
}
catch (Exception ex)
{
   Console.WriteLine("Allgemeiner Fehler: {0}", ex.Message);
}
```

Listing 280: Anwendung der Methode zum Extrahieren eines Zip-Archivs

XML

169 XML-Dokumente über LINQ to XML lesen

LINQ (Language Integrated Query) ermöglicht in Form von *LINQ To XML* das Lesen (und in Form des X-DOM auch das Schreiben, siehe Rezept 179) von XML-Dokumenten. Ein Vorteil gegenüber dem Lesen über einen `XmlReader` (Rezept 171) oder das W3C-DOM ist, dass Sie mit LINQ eine Abfragesprache verwenden, die Sie mit derselben Syntax auch zur Abfrage von Auflistungen und Datenbanken verwenden können. Ein anderer Vorteil ist ein vereinfachtes Verarbeiten von XML-Dokumenten, ohne auf spezielle XML-Features wie XPath und XSLT zurückgreifen zu müssen.

Hinweis

Beachten Sie auch den Artikel »LINQ« im Ordner *Zusatz-Artikel* auf der Buch-CD, der die LINQ-Grundlagen recht ausführlich erläutert.

Im Vergleich mit einem `XmlReader` sollte LINQ to XML eine ähnlich gute Performance ergeben (da intern ein `XmlReader` verwendet wird), LINQ to XML ermöglicht aber eine höhere Produktivität und erweiterte Features. Microsoft schreibt, dass Sie zum Lesen von XML-Dokumenten immer dann einen `XmlReader` verwenden sollten, wenn Sie viele gleichartige Dokumente verarbeiten müssen und dazu prinzipiell dieselbe Logik (z. B. in Form einer Methode) verwenden können. LINQ to XML werden Sie wahrscheinlich dann verwenden, wenn die zu bearbeitenden XML-Dokumente relativ klein und relativ unterschiedlich sind.

Eines der erweiterten Features, das ein `XmlReader` oder ein `XmlDocument` nicht und ein `XslCompiledTransform`-Objekt lediglich über das relativ komplexe XSLT bietet (siehe Rezept 186), sind LINQ to XML Transformationen, die das Transformieren von XML-Dokumenten auf eine flexible und leicht lesbare Art ermöglichen. LINQ to XML Transformationen werden im Rezept 185 beschrieben.

Das X-DOM

LINQ to SQL basiert auf einem Objektmodell, das dem W3C-DOM ähnlich ist. Die entsprechenden Klassen werden im Namensraum `System.Xml.Linq` verwaltet. Die wichtigsten sind `XElement`, `XDocument` und `XNode`. Diese Klassen bilden ein speziell auf LINQ abgestimmtes DOM, aber nicht das W3C-DOM (dieses wird von älteren Klassen aus dem Namensraum `System.Xml` wie `XmlDocument` und `XmlElement` implementiert). In Ermangelung eines offiziellen Begriffs nenne ich das LINQ-to-XML-DOM »X-DOM«.

Sie können das X-DOM separat von LINQ to SQL verwenden, wie die Rezepte 172 und 179 zeigen. Das X-DOM ist aber für LINQ to XML optimiert (und unterscheidet sich deswegen auch vom W3C-DOM). So geben viele der X-DOM-Methoden eine `IEnumerable<T>`-Auflistung zurück, die mit LINQ abgefragt werden kann. Aus dieser Sicht betrachtet besteht LINQ to XML lediglich aus einigen speziell auf das X-DOM abgestimmten Erweiterungsmethoden.

XML-Dokumente ohne Namensraum lesen

Zur Abfrage eines einfachen (Namensraum-losen) XML-Dokuments über LINQ benötigen Sie eine Instanz der Klasse `XDocument` oder der Klasse `XElement`. `XDocument` repräsentiert ein komplettes XML-Dokument, `XElement` ein XML-Element (inkl. Kind-Elementen). `XElement` kann prinzipiell auch ein XML-Dokument repräsentieren, da es sich bei einem XML-Dokument ja im Prinzip nur um das Wurzel- Element eines XML-Dokuments mit seinen Kind-Elementen handelt. Allerdings bietet `XElement` nicht die Möglichkeit, Top-Level-Konstrukte wie Kommentare oder Verarbeitungs-Instruktionen zu verarbeiten.

Bei der Abfrage der Daten können Sie verschiedene Methoden und Eigenschaften der `XContainer`-Klasse verwenden, von der `XElement` und `XDocument` abgeleitet sind. Die (für die Abfrage) wichtigsten werden in Tabelle 19 beschrieben.

Methode	Beschreibung
`XElement Element (XName name)`	liefert das erste Kind-Element mit einem Namen, der dem übergebenen `XName`-Objekt entspricht. Existiert kein Kind-Element mit dem übergebenen Namen, wird `null` zurückgegeben. Ein `XName`-Objekt verwaltet einen Namen und optional einen XML-Namensraum. Wenn Sie nur einen Namen angeben wollen, können Sie einen einfachen String einsetzen, der dann implizit in ein `XName`-Objekt konvertiert wird.
`XAttribute Attribute (XName name)`	gibt ein `XAttribute`-Objekt für das (erste) Attribut zurück, das den angegebenen Namen trägt.
`IEnumerable<XAttribute> Attributes([XName name])`	liefert eine Auflistung der Attribute des XML-Elements. Wenn Sie einen Namen übergeben, werden nur die Attribute zurückgeliefert, die dem Namen entsprechen.
`IEnumerable<XElement> Ancestors([XName name])`	liefert alle Vorfahren des XML-Elements (auch die in höheren Ebenen als der direkte Vorfahre). Wenn Sie einen Namen übergeben, werden nur die Vorfahren zurückgeliefert, die den angegebenen Namen tragen.
`IEnumerable<XElement> Descendants ([XName name])`	liefert die Nachfahren (Kinder) des XML-Elements (auch die in tieferen Ebenen als die direkten Nachfahren). Sie können dieser Methode einen Namen übergeben, um nur die Elemente zu erhalten, die den angegebenen Namen tragen.
`IEnumerable<XElement> ElementsAfterSelf()`	liefert die Elemente auf derselben Ebene (die »Geschwister«), die hinter dem Element liegen.
`IEnumerable<XElement> ElementsBeforeSelf()`	liefert die Elemente auf derselben Ebene (die »Geschwister«), die vor dem Element liegen.
`XName Name`	gibt den Namen des XML-Elements zurück.
`XElement Parent`	referenziert das Vater-Element des XML-Elements, sofern dieses ein solches besitzt (das Wurzel-Element besitzt natürlich kein Vater-Element, `Parent` ergibt hier `null`).
`string Value`	verwaltet den Wert des XML-Elements, falls dieses einen Wert besitzt.

Tabelle 19: Die wichtigsten Methoden und Eigenschaften der XElement-Klasse

Als Beispiel soll die folgende XML-Datei eingelesen werden, die Personendaten verwaltet:

```
<?xml version="1.0" encoding="utf-8" ?>
<persons>
  <person id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <type>Alien</type>
  </person>

  <person id="1001">
    <firstname>Ford</firstname>
    <lastname>Prefect</lastname>
    <type>Alien</type>
  </person>

  <person id="1002">
    <firstname>Tricia</firstname>
    <lastname>McMillan</lastname>
    <type>Earthling</type>
  </person>

  <person id="1003">
    <firstname>Arthur</firstname>
    <lastname>Dent</lastname>
    <type>Earthling</type>
  </person>
</persons>
```

Listing 281: Beispiel-XML-Datei mit den Daten von Personen

Das Beispiel liest alle Personen ein, deren Typ »Alien« ist. Die Daten der Personen werden in konstruierte anonyme Typen eingelesen und nach dem Einlesen in einer Schleife ausgegeben. Zum Kompilieren dieses Beispiels müssen Sie die Assemblys *System.dll*, *System.Core.dll*, *System.Xml.dll* und *System.Xml.Linq.dll* referenzieren und die entsprechenden Namensräume importieren. Das Beispiel erfordert zudem den Import der Namensräume `System.IO` und `System.Reflection`.

```
// Dateiname der XML-Datei zusammenstellen
string xmlFileName = "Persons.xml";

// XElement zum Lesen erzeugen
XElement personsXml = XElement.Load(xmlFileName);
// Alternative: XDocument (inkl. Möglichkeiten,
// Top-Level-Features zu verarbeiten)
```

Listing 282: Einlesen eines XML-Dokuments ohne Namensraum über LINQ to XML

```
// XDocument personsXml = XDocument.Load(xmlFileName);

// Alle Aliens abfragen
var persons =
   from person in personsXml.Descendants("person")
   where person.Element("type").Value == "Alien"
   select new
      {
         Id = (int)person.Attribute("id"),
         FirstName = person.Element("firstname").Value,
         LastName = person.Element("lastname").Value
      };

// Die abgefragten Personen durchgehen und deren Daten ausgeben
foreach (var person in persons)
{
   Console.WriteLine(person.Id + ": " +
      person.FirstName + " " + person.LastName);
}
```

Listing 282: Einlesen eines XML-Dokuments ohne Namensraum über LINQ to XML (Forts.)

Das Beispiel fragt alle Nachfahren des Wurzel-Elements des XML-Dokuments ab, deren Name »person« ist und deren Kind-Element »type« den Wert »Alien« speichert. Der Personen-Typ wird über die `Element`-Methode ausgewertet, die ein `XElement` für das Kind-Element mit dem übergebenen Namen (ohne Namensraum) zurückgibt. Die Daten der ermittelten Personen werden in eine Instanz eines anonymen Typs geschrieben, wobei die `Attribute`-Methode verwendet wird, um den Wert des *id*-Attributs zu ermitteln. Das Beispiel nutzt hier die Tatsache, dass `XAttribute`-(und `XElement`-)Objekte explizit in die Standard-Typen konvertiert werden können. Der Vor- und der Nachname der Person werden dann über die Element-Methode ausgelesen. Das Beispiel fragt hier die `Value`-Eigenschaft ab, könnte das zurückgegebene `XElement` aber auch mit `(string)` in einen String konvertieren. Schließlich werden die abgefragten Personen dann nur noch ausgegeben.

170 XML-Dokumente mit Namensraum über LINQ to XML lesen

XML-Dokumente sind häufig mit Namensräumen ausgestattet. Diese müssen Sie beim Lesen (und beim Durchgehen und Erzeugen) berücksichtigen.

Das Beispiel dieses Rezepts demonstriert, wie ein XML-Dokument eingelesen wird, dessen Elemente einem XML-Namensraum zugeordnet sind.

Exkurs

XML-Namensräume haben eine ähnliche Bedeutung wie die Namensräume in .NET: Sie trennen Elemente auf einer übergeordneten Ebene voneinander. Ein Element *A*, das dem Namensraum *X* zugeordnet ist, ist ein vollkommen anderes Element als ein Element *A*, das dem Namensraum *Y* zugeordnet ist. Bedeutung haben XML-Namensräume beim Zusammenführen von verschiedenen XML-Dokumenten. Dabei kann es vorkommen, dass die zusammengeführten Dokumente auf derselben Ebene gleichnamige Elemente beinhalten. Gehören diese unterschiedlichen Namensräumen an, ist das aber kein Problem, da die Elemente über ihren Namensraum adressiert werden.

XML-Namensräume können alles Mögliche sein. In der Praxis werden häufig URIs und GUIDs verwendet. URIs müssen dabei nicht auf eine wirklich existierende Ressource im Internet verweisen, sondern können vollkommen fiktiv sein.

Zum Hinzufügen von Namensräumen gibt es zwei Möglichkeiten: Die einfachste ist, einem übergeordneten Element (in der Regel ist das das Root-Element) über das `xmlns`-Attribut einen Namensraum zuzuordnen:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<persons xmlns="http://www.addison-wesley.de/codebook">
   <person id="1000">
      ...
   </person>
</persons>
```

In diesem Fall werden alle untergeordneten Elemente automatisch ebenfalls dem angegebenen Namensraum zugeordnet. Das `person`-Element im Beispiel gehört also genau wie das `persons`-Element dem Namensraum *http://www.addison-wesley.de/codebook* an.

Die andere Möglichkeit ist, bei der Deklaration des Namensraums einen Präfix anzugeben und mit diesem alle Elemente zu kennzeichnen, die dem Namensraum zugeordnet werden sollen:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<awc:persons xmlns:awc="http://www.addison-wesley.de/codebook">
   <awc: person id="1000">
      ...
   </awc: person>
</awc:persons>
```

Diese Variante ist deutlich schwieriger und fehleranfälliger und sollte nur dann angewendet werden, wenn in einem XML-Dokument mit mehreren Namensräumen gearbeitet wird. Ein typischer Fehler wäre z. B. untergeordnete Elemente nicht mit dem Präfix zu versehen:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<awc:persons xmlns:awc="http://www.addison-wesley.de/codebook">
   <person id="1000">
      ...
   </person>
</awc:persons>
```

In diesem Beispiel ist nur das Element `persons` dem Namensraum zugeordnet, `person` gehört keinem Namensraum an. Das gibt natürlich dann Probleme, wenn Sie über XPath nach Elementen suchen.

Das XML-Dokument für dieses Beispiel sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8" ?>
<cars xmlns="http://www.addison-wesley.com/codebook">
  <car>
    <make>Citroen</make>
    <model>C1</model>
  </car>
  <car>
    <make>Ford</make>
    <model>Puma</model>
  </car>
</cars>
```

Listing 283: Beispiel-XML-Dokument mit Namensraum

Der Bezug auf ein Kind-Element oder Attribut erfolgt bei LINQ to XML über ein `XName`-Objekt. Ein solches Objekt verwaltet den Namen des Elementes oder des Attributs und den Namensraum. Per Voreinstellung ist der globale Namensraum (ein leerer String) eingestellt.

`XName` ist in Bezug auf Konvertierungen sehr flexibel. Ein String, den Sie einer Methode als Elementname übergeben, wird von einem Konvertierungsoperator der `XName`-Klasse in eine `XName`-Instanz umgesetzt. Eine Besonderheit ist nun, dass Sie den Namensraum im String angeben können, indem Sie diesen in geschweifte Klammern einschließen:

```
"{http://www.addison-wesley.com/codebook}car")
```

Eine bessere Möglichkeit ist aber, den Namensraum zuvor in einem `XNamespace`-Objekt zu verwalten (dem Sie aber auch einen einfachen String zuweisen können) und eine Addition aus diesem Objekt mit dem eigentlichen Namen zu verwenden. Der überladene Additions-Operator von `XNamespace` erzeugt aus der Addition eines `XNamespace`-Objekts mit einem String ein `XName`-Objekt, das entsprechend initialisiert ist.

So können Sie sehr einfach auf Elemente zugreifen, die einem Namensraum zugeordnet sind:

```
// XML-Dokument mit Namensraum einlesen
string xmlFileName = "Cars.xml";
XElement carsXml = XElement.Load(xmlFileName);

// XNamespace-Objekt für den Namensraum erzeugen
XNamespace ns = "http://www.addison-wesley.com/codebook";

// Das XML-Dokument abfragen
var cars =
   from car in carsXml.Elements(ns + "car")
   select new
   {
```

Listing 284: Lesen eines XML-Dokuments mit Namensraum

```
      Make = car.Element(ns + "make").Value,
      Model = car.Element(ns + "model").Value
   };

// Das Ergebnis ausgeben
Console.WriteLine("XML-Dokument mit Namensraum:");
foreach (var car in cars)
{
   Console.WriteLine(car.Make + " " + car.Model);
}
```

Listing 284: Lesen eines XML-Dokuments mit Namensraum (Forts.)

Hinweis

Das Einlesen eines XML-Dokuments mit der Präfix-Namensraum-Syntax sieht prinzipiell genauso aus. Hierbei müssen Sie natürlich darauf achten, dass einzelne Elemente oder Attribute unterschiedlichen Namensräumen zugeordnet sein können.

171 XML-Dokumente über ein XmlReader-Objekt lesen

Ein XML-Dokument können Sie neben LINQ to XML (Rezept 169) und dem X-DOM (Rezept 172) auch über eine Instanz der Klasse `System.Xml.XmlReader` lesen. Ein solches Objekt erlaubt ein einmaliges, schreibgeschütztes Durchgehen der einzelnen XML-Knoten (unabhängig von jeglichen Namensräumen). Da dabei keine Daten im Arbeitsspeicher gehalten werden, benötigt diese Klasse sehr wenig Ressourcen und ermöglicht ein schnelles Lesen. Wenn Sie mehrere gleichartige XML-Dokumente mit demselben Programmcode nur sequenziell lesen und nicht in diesen Daten suchen wollen, ist ein `XmlReader`-Objekt die erste Wahl (ansonsten verwenden Sie LINQ to XML oder die (X-)DOM-Variante).

Hinweis

Die Verwendung einer `XmlReader`-Instanz hat übrigens viel Ähnlichkeit mit *SAX* (Simple API for XML). Der Unterschied ist, dass Sie bei der Verwendung eines `XmlReader`-Objekts die Daten über Methoden explizit auslesen (was als »Pull« bezeichnet wird). SAX hingegen geht ein XML-Dokument intern durch und informiert das Programm über Ereignisse, dass ein neuer XML-Knoten erreicht wurde (»Push«). Wahrscheinlich ist keine dieser Methoden besser als die andere. Auf jeden Fall sind beide ressourcenschonend. SAX wird vom .NET Framework nicht unterstützt. Sie können jedoch spezielle XML-Komponenten wie Microsofts MSXML-Parser einsetzen, wenn Sie SAX verwenden wollen.

Zum Lesen einer XML-Datei müssen Sie zunächst eine Instanz der `XmlReader`-Klasse (aus dem Namensraum `System.Xml`) erzeugen. Dazu verwenden Sie die statische `Create`-Methode der `XmlReader`-Klasse, der Sie u. a. den Dateinamen einer XML-Datei oder einen URL angeben können, der eine XML-Datei im Internet bezeichnet:

```
xmlReader = XmlReader.Create(xmlFileName);
```

Nun rufen Sie die `Read`-Methode in einer Schleife auf und verarbeiten die XML-Daten im Schleifenkörper:

```
while (xmlReader.Read())
{
   // Die XML-Daten verarbeiten
}
```

`Read` setzt das `XmlReader`-Objekt auf den jeweils nächsten Knoten. Die Daten dieses Knoten können Sie dann über Methoden und Eigenschaften des `XmlReader`-Objekts lesen.

Bei der Verarbeitung müssen Sie die verschiedenen XML-Knoten unterscheiden. XML kennt einige XML-Knotentypen, wie z. B. einen Start-Tag, die Textdaten eines Elements oder einen End-Tag. Wenn Sie eine XML-Datei gezielt durchgehen wollen, müssen Sie die Struktur der XML-Daten kennen und beim Durchgehen der einzelnen Knoten deren Typ und Namen abfragen. Den Typ eines Knotens erhalten Sie über die Eigenschaft `NodeType` als Wert der Aufzählung `XmlNodeType`. Die wichtigsten `XmlNodeType`-Konstanten sind `Element` (Start-Tag), `EndElement` (Ende-Tag), `Text` (der Text eines Elements) und `Attribute` (ein Attribut). Den Namen des aktuellen Knotens können Sie aus der Eigenschaft `Name` des `XmlReader`-Objekts auslesen, den Wert erhalten Sie über die Eigenschaft `Value`.

Das Beispielprogramm soll alle in der XML-Datei aus dem Rezept 169 gespeicherten Personen in Instanzen einer eigenen Klasse `Person` einlesen, die in einer `List`-Auflistung verwaltet werden. Ich habe diesen Weg gewählt, um möglichst viel Praxisnähe in das Programm zu integrieren. Die Personen-Auflistung kann dann nach dem Lesen weiterverarbeitet werden. Das Beispiel gibt die eingelesenen Daten allerdings einfach nur an der Konsole aus.

Das Programm geht zum Einlesen der Personen die einzelnen Knoten der XML-Datei durch und überprüft zunächst, ob eine neue Person beginnt. Ist dies der Fall, erzeugt es eine neue Instanz der Klasse `Person`. Über die Methode `GetAttribute` wird dann das Attribut `id` des `person`-Elements eingelesen und im `Person`-Objekt gespeichert.

Für die folgenden Element-Knoten überprüft das Programm, um welches Element es sich handelt. Handelt es sich um einen der Start-Tags der Elemente, die die Personendaten verwalten, wechselt es über die `MoveToContent`-Methode zum nächsten Inhalts-Knoten, der in unserem Fall ein Text-Knoten ist. `MoveToContent` überspringt Kommentare und andere, für die eigentlichen Daten unwichtige Knoten und gibt den Knotentyp zurück, wenn (noch) ein Inhaltsknoten gefunden wurde. Über den Vergleich auf `XmlNodeType.None` wird überprüft, ob eventuell kein Inhaltsknoten mehr gefunden wurde.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Xml`, `System.IO`, `System.Collections.Generic` und `System.Windows.Forms` einbinden.

```
/* Klasse zur Speicherung der Daten einer Person */
class Person
{
   public string Id;
   public string FirstName;
   public string LastName;
```

Listing 285: Einlesen einer XML-Datei über einen XmlReader

```
   public string Type;
}

/* Start-Klasse */
class Program
{
   [STAThread]
   static void Main(string[] args)
   {
      Console.Title = "XML-Dateien über ein XmlReader-Objekt lesen";

      // Der Dateiname
      string xmlFileName = Path.Combine(Application.StartupPath,
         "Persons.xml");

      // XmlReader erzeugen
      XmlReader xmlReader = null;
      try
      {
         xmlReader = new XmlReader(xmlFileName);
      }
      catch (Exception ex)
      {
         MessageBox.Show("Fehler beim Einlesen der XML-Datei '" +
            xmlFileName + "': " + ex.Message, Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Error);
         return;
      }

      // Auflistung für die Personen erzeugen
      List<Person> persons = new List<Person>();

      // Die einzelnen Knoten einlesen und durchgehen
      Person person = null;
      try
      {
         while (xmlReader.Read())
         {
            // Überprüfen, ob es sich beim aktuellen Knoten um einen
            // Start-Tag handelt
            if (xmlReader.NodeType == XmlNodeType.Element)
            {
               // Überprüfen, ob es sich um ein person-Element handelt
               if (xmlReader.Name == "person")
               {
```

Listing 285: Einlesen einer XML-Datei über einen XmlReader (Forts.)

```
                // Neue Person: Neues Person-Objekt erzeugen ...
                person = new Person();

                // ... und in der Auflistung ablegen
                persons.Add(person);

                // Das Attribut 'id' einlesen
                person.Id = xmlReader.GetAttribute("id");
            }
            else
            {
                // Überprüfen, um welches Element es sich handelt
                if (xmlReader.Name == "firstname")
                {
                    // Zum Inhalt wechseln und den Text einlesen
                    if (xmlReader.MoveToContent() != XmlNodeType.None)
                    {
                        person.FirstName = xmlReader.ReadString();
                    }
                }
                else if (xmlReader.Name == "lastname")
                {
                    // Zum Inhalt wechseln und den Text einlesen
                    if (xmlReader.MoveToContent() != XmlNodeType.None)
                    {
                        person.LastName = xmlReader.ReadString();
                    }
                }
                else if (xmlReader.Name == "type")
                {
                    // Zum Inhalt wechseln und den Text einlesen
                    if (xmlReader.MoveToContent() != XmlNodeType.None)
                    {
                        person.Type = xmlReader.ReadString();
                    }
                }
            }
        }
    }
}
catch (Exception ex)
{
    MessageBox.Show("Fehler beim Einlesen der XML-Datei '" +
        xmlFileName + "': " + ex.Message, Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
```

Listing 285: Einlesen einer XML-Datei über einen XmlReader (Forts.)

```
            xmlReader.Close();
            return;
        }

        // XmlReader schließen
        xmlReader.Close();

        // Alle eingelesenen Personen durchgehen und an der Konsole ausgeben
        foreach (Person p in persons)
        {
            Console.WriteLine("Person {0}", p.Id);
            Console.WriteLine("Vorname: {0}", p.FirstName);
            Console.WriteLine("Nachname: {0}", p.LastName);
            Console.WriteLine("Typ: {0}", p.Type);
            Console.WriteLine();
        }
    }
}
```

Listing 285: Einlesen einer XML-Datei über einen XmlReader (Forts.)

Hinweis

Entsprechend der XML-Empfehlung des *W3C*[1] sollten Sie das Lesen einer XML-Datei komplett abbrechen, wenn diese fehlerhaft ist. Nur so genannte wohlgeformte (und damit fehlerfreie) XML-Dateien sollten demnach verarbeitet werden.

172 XML-Dokumente über das X-DOM lesen

Wenn Sie beim Einlesen von XML-Dateien gezielt auf einzelne Elemente zugreifen und dabei vermeiden wollen, das gesamte Dokument sequenziell durchgehen zu müssen, können Sie ein `XElement`- oder `XDocument`-Objekt zum Lesen verwenden. Ein solches Objekt erlaubt über verschiedene Methoden den wahlfreien lesenden und schreibenden Zugriff auf ein XML-Dokument (und nebenbei den sehr flexiblen Zugriff über LINQ to XML, siehe Rezept 169).

`XElement` und `XDocument` gehören zu der Microsoft-eigenen DOM-Implementierung, die speziell für LINQ to XML entwickelt wurde und die ich (und andere Autoren) X-DOM nenne. Der Begriff »DOM« (Document Object Model) stammt ursprünglich vom *W3C*. Das *World Wide Web Consortium* (*www.w3.org*) entwickelt und empfiehlt Technologien und Standards für das Internet, wie z. B. HTML, Cascading Style Sheets, XML und eben das DOM.

Das W3C-DOM legt einen Satz an Klassen und deren Methoden und Eigenschaften fest, die ein XML-Dokument im Arbeitsspeicher repräsentieren und über die dieses Dokument gelesen und bearbeitet werden kann. Das X-DOM ist ähnlich aufgebaut, aber wesentlich performanter und flexibler als die alte W3C-DOM-Implementierung von Microsoft (in Form der Klassen `XmlDocument`, `XmlElement` etc.).

1. Das World Wide Web Consortium (W3C, *www.w3.org*) entwickelt und empfiehlt Technologien und Standards für das Internet, wie z. B. HTML, XML und Cascading Style Sheets.

Über diese Klassen können Sie ein XML-Dokument sehr flexibel bearbeiten. Das Lesen und Schreiben verbraucht aber im Vergleich zum SAX-ähnlichen Lesen mit einem `XmlReader`-Objekt (das ich im Rezept 171 beschrieben habe) und zum Schreiben mit einer `XmlWriter`-Instanz mehr Arbeitsspeicher, da das XML-Dokument komplett in den Speicher gelesen und dort verarbeitet wird. Das sequenzielle Lesen eines XML-Dokuments ist über das X-DOM auch langsamer als mit den SAX-ähnlichen Klassen. Dafür können Sie gezielt auf XML-Elemente zugreifen, diese lesen, aktualisieren, löschen und neue Elemente hinzufügen.

Die zum Lesen eines kompletten XML-Dokuments oder eines Teilbereichs daraus wichtigsten Methoden sind `Element` und `Elements`. `Element` übergeben Sie den Namen eines Kind-Elements. Falls ein Element mit diesem Namen existiert, erhalten Sie eine (`XElement`-)Referenz darauf zurück (im anderen Fall ist die Rückgabe `null`). `Elements` liefert alle Kind-Elemente. Dieser Methode können Sie einen Namen übergeben, um nur die Elemente auszulesen, die diesen Namen tragen.

Für unser XML-Beispiel aus dem Rezept 171 rufe ich in Listing 286 `Elements` für das Wurzel-Element des XML-Dokuments auf, um alle Kind-Elemente zu erhalten, die Personendaten speichern. Listing 286 speichert zur Demonstration alle gelesenen Personen in `Person`-Instanzen und fügt diese einer Auflistung an.

Hinweis

Dieses Problem könnten Sie viel eleganter über eine Projektion lösen, wie ich in Rezept 169 zeige. Dieses Rezept soll aber grundlegend zeigen, wie Sie auch komplexe XML-Dokumente quasi sequenziell durchgehen können.

Die Klassen `XNode`, `XContainer` und `XElement` sind die wichtigsten Klassen des X-DOM. `XNode` repräsentiert einen Knoten (der alles sein kann, was in einem XML-Dokument vorkommt, zum Beispiel auch ein Kommentar). `XContainer` steht für einen Knoten, der Kind-Elemente besitzen kann. `XElement` ist schließlich ein XML-Element (mit Start- und Ende-Tag und Inhalt). `XElement` ist von `XContainer` und `XContainer` von `XNode` abgeleitet.

Zahlreiche Eigenschaften und Methoden erlauben das flexible Arbeiten mit den entsprechenden Objekten. Die `XNode`-Eigenschaft `Parent` liefert z. B. eine Referenz auf den übergeordneten Knoten, die Methode `XContainer.Elements` liefert Referenzen auf alle untergeordneten Elemente in Form einer Auflistung, die Eigenschaft `XContainer.FirstNode` repräsentiert den ersten Unterknoten (falls ein solcher existiert).

Über die `Attribute`-Methode eines `XElement`-Objekts erhalten Sie Zugriff auf die Attribute eines Elements. Die `Element`-Methode ermöglicht die gezielte Auswahl eines Unterelements. Im Erfolgsfall liefert diese Methode ein `XmlElement`-Objekt zurück, das das Unterelement repräsentiert. Wird kein Unterelement gefunden, erhalten Sie `null` zurück. Für unser Beispiel rufe ich diese Methode unter Übergabe der Tagnamen für die Unterelemente des `person`-Elements auf, um die Eigenschaften einer Person gezielt einzulesen. Den in einem Element gespeicherten Text lese ich dann über die Eigenschaft `Value` ein.

Das Beispiel aus dem Rezept 171, das die einfache Personen-XML-Datei (ohne Namensraum) in eine `List<Person>`-Auflistung einliest, sieht mit dem X-DOM programmiert so aus wie in Listing 286.

Hinweis

Falls das XML-Dokument oder Teile davon einem Namensraum zugeordnet ist/sind, müssen Sie dieselbe Technik anwenden wie in Rezept 170.

Das Programm benötigt den Import der Namensräume `System`, `System.Collections.Generic` und `System.Xml.Linq`.


```
// Der Dateiname
string xmlFileName = "Persons.xml";

// Das Dokument in eine neue XElement-Instanz laden
XElement rootElement = XElement.Load(xmlFileName);

// Auflistung für die Personen erzeugen
List<Person> persons = new List<Person>();

// Alle person-Elemente einlesen und durchgehen
var personElements = rootElement.Elements("person");
foreach (var personElement in personElements)
{
   // Neue Person erzeugen und in der Auflistung ablegen
   Person person = new Person();
   persons.Add(person);

   // Das Attribut id einlesen
   XAttribute idAttribute = personElement.Attribute("id");
   if (idAttribute != null)
   {
      person.Id = idAttribute.Value;
   }

   // firstname-Element suchen und speichern
   XElement firstNameElement = personElement.Element("firstname");
   if (firstNameElement != null)
   {
      person.FirstName = firstNameElement.Value;
   }

   // lastname-Element suchen und speichern
   XElement lastNameElement = personElement.Element("lastname");
   if (lastNameElement != null)
   {
      person.LastName = lastNameElement.Value;
   }
```

Listing 286: Quasi-sequenzielles Einlesen einer XML-Datei über das X-DOM

```
   // type-Element suchen und speichern
   XElement typeElement = personElement.Element("type");
   if (typeElement != null)
   {
      person.Type = typeElement.Value;
   }
}

// Alle eingelesenen Personen durchgehen und an der Konsole ausgeben
foreach (Person p in persons)
{
   Console.WriteLine("Person {0}", p.Id);
   Console.WriteLine("Vorname: {0}", p.FirstName);
   Console.WriteLine("Nachname: {0}", p.LastName);
   Console.WriteLine("Typ: {0}", p.Type);
   Console.WriteLine();
}
```

Listing 286: Quasi-sequenzielles Einlesen einer XML-Datei über das X-DOM (Forts.)

Hinweis

Sie sollten beachten, dass das in diesem Rezept vorgestellte Einlesen über ein `XElement`- oder `XDocument`-Objekt die XML-Daten nicht gegen ein eingebettetes oder externes DTD oder Schema überprüft. Um dies zu erreichen, lesen Sie das Rezept 175.

173 XML-Dokumente beim Einlesen gegen ein Schema (oder DTD) prüfen

XML-Schemas legen die Struktur eines XML-Dokuments fest. Unter anderem kann in einem Schema der Datentyp von XML-Elementen festgelegt oder definiert sein, dass und wie oft bestimmte Elemente unterhalb anderer Elemente vorkommen müssen bzw. dürfen.

Hinweis

DTD (Document Type Definition) ist eine ältere Sprache zur Definition der Struktur von XML-Dokumenten. XML-Schemata sind die zurzeit aktuelle Variante. Schemata erlauben wesentlich mehr Festlegungen, wie z. B. die Bestimmung des genauen Datentyps von XML-Elementen, was bei DTDs nicht möglich ist. DTDs und Schemata können in XML-Dokumenten eingebettet sein oder in externen Dateien vorliegen. Extern vorliegende Schemata oder DTDs können in einem XML-Dokument über festgelegte XML-Instruktionen mit dem Dokument verbunden werden. Schauen Sie sich die XML-Dateien im Beispiel zum Rezept 174 an, wenn Sie sehen wollen, wie dies in der Praxis aussieht.

Tipp

Hervorragende Informationen zu DTDs finden Sie bei den *W3Schools* an der Adresse *www.w3schools.com/dtd/default.asp*, Schemata werden an der Adresse *www.w3schools.com/schema/default.asp* behandelt.

Ein Schema für die Personen-XML-Datei aus dem Rezept 171 könnte z. B. folgendermaßen aussehen:

```
<?xml version="1.0" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" id="persons">
   <xs:element name="persons">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="person" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:all>
                     <xs:element name="firstname" type="xs:string" />
                     <xs:element name="lastname" type="xs:string" />
                     <xs:element name="type" type="xs:string" />
                  </xs:all>
                  <xs:attribute name="id" type="xs:int" use="required" />
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Listing 287: Beispielschema für die Personen-XML-Datei

Das Beispielschema legt fest, dass das Wurzel-Element `persons` ein komplexer Typ (mit Unterelementen) ist, der beliebig viele `person`-Elemente enthalten kann. Ein `person`-Element ist ebenfalls als komplexer Typ definiert, der die Elemente `firstname`, `lastname` und `type` genau ein Mal enthalten muss. Für die einzelnen Unterelemente ist jeweils der Datentyp festgelegt. Schließlich wird noch definiert, dass das Element `person` ein Attribut besitzen muss, das `id` heißt und das den (XML-)Typ `int` besitzt.

Um eine XML-Datei beim Einlesen gegen ein Schema zu prüfen, können Sie wie beim normalen Einlesen (Rezept 171) eine Instanz der Klasse `XmlReader` verwenden. Die Eigenschaft `Settings` verwaltet Einstellungen, die beim Lesen berücksichtigt werden sollen. `Settings` ist vom Typ `XmlReaderSettings`. Die Eigenschaft `ValidationType` dieser Klasse bestimmt die Art der Validierung in Form der Werte der `ValidationType`-Aufzählung. Die einzelnen Werte dieser Aufzählung erläutere ich in Rezept 174. Damit können Sie die eingelesenen Daten gegen ein DTD (Document Type Definition), gegen ein XML-Schema oder einfach nur auf Wohlgeformtheit prüfen. Wenn Sie gegen ein XML-Schema prüfen wollen, übergeben Sie hier den Wert `ValidationType.Schema`.

Ist das Schema nicht in der XML-Datei eingebettet, müssen Sie dieses der `Schemas`-Auflistung des `XmlReaderSettings`-Objekts hinzufügen. Dazu verwenden Sie deren `Add`-Methode, der Sie eine vorhandene `XmlSchemaCollection`-Instanz, ein vorhandenes `XmlSchema`-Objekt, einen `XmlReader` oder einfach nur den Dateinamen oder die URI (z. B. die Webadresse) einer Schemadatei übergeben können. In den letzten beiden Fällen können Sie am ersten Argument einen XML-

Namensraum-URI[2] übergeben. Namensräume werden in XML verwendet, um XML-Elemente (die einem bestimmten Namensraum zugeordnet sind) von anderen XML-Elementen (die anderen Namensräumen zugeordnet sind) eindeutig zu unterscheiden. Der Namensraum, den Sie am ersten Argument der Methode `Schemas.Add` übergeben, ist der, der im Attribut `targetNamespace` der Schema-Datei angegeben ist (falls dieses vorhanden ist). Ist ein solcher Ziel-Namensraum angegeben, müssen die Elemente der zu validierenden XML-Datei diesem Namensraum zugeordnet sein. Damit wird sichergestellt, dass ein XML-Schema nur für bestimmte XML-Namensräume verwendet werden kann. Ist dieses Attribut nicht vorhanden, geben Sie einfach `null` an.

Dummerweise referenziert die `Settings`-Eigenschaft einer neu erzeugten `XmlReader`-Instanz noch kein `XmlReaderSettings`-Objekt. Sie können also leider nicht einfach eine `XmlReader`-Instanz erzeugen und `Settings` initialisieren. Um das Ganze noch ein wenig komplizierter zu machen, bietet die `XmlReader`-Klasse auch keine Möglichkeit, ein `XmlReaderSettings`-Objekt im Konstruktor zu übergeben. Lediglich die (von `XmlReader` geerbte) statische `Create`-Methode ermöglicht die Übergabe eines solchen Objekts. Allerdings bietet diese Methode keine Möglichkeit, lediglich den Pfad zur XML-Datei anzugeben. Sie können allerdings u. a. ein `FileStream`-Objekt übergeben, das die XML-Datei repräsentiert. `Create` gibt eine `XmlReader`-Referenz zurück, die Sie nicht nach `XmlReader` casten können. Sie können die Daten aber problemlos auch über diese Referenz einlesen und überprüfen.

Hinweis

Problematisch wird die Verwendung einer `XmlReader`-Instanz zum Validieren eines XML-Dokuments über ein Schema, das (u. U. auch nur teilweise) über das Internet abgerufen wird, auf einem Rechner, der über einen Proxy-Server Verbindung zum Internet erhält. Leider können Sie einer `XmlReader`-Instanz scheinbar keine direkten Informationen für den Proxy-Zugang übergeben. Das Schema kann in diesem Fall nicht eingelesen werden, da Sie weder die Proxy-Adresse noch die Zugangsdaten angeben können. Eine Lösung dieses Problems bietet die `XmlResolver`-Eigenschaft der `XmlReaderSettings`-Klasse, wie ich es in Rezept 173 zeige.

Wenn Sie nun eine `XmlReader`-Instanz zum Lesen verwenden, erzeugt diese beim Lesen eine Ausnahme vom Typ `XmlException`, wenn die eingelesenen Daten nicht wohlgeformt sind oder nicht dem Schema entsprechen.

Hinweis

Im nächsten Rezept stelle ich eine Klasse zum allgemeinen Validieren von XML-Dokumenten vor. Dort beschreibe ich, dass Sie auch dem `Validation`-Ereignis der `XmlReader`-Instanz eine passende Ereignisbehandlungsmethode zuweisen können. In diesem Fall erzeugt das `XmlReader`-Objekt bei Fehlern im Dokument keine Ausnahme, sondern ruft die zugewiesene Methode auf. Damit können Sie erreichen, dass das Lesen der XML-Datei nicht beim ersten Fehler abgebrochen wird, und somit alle Fehler eines Dokuments auswerten.

2. Ein URI (Uniform Ressource Identifier) identifiziert Ressourcen (HTML-Dokumente, FTP-Dateien, Mailboxen etc.) im Internet. Die Adresse eines HTML-Dokuments oder die Adresse einer Mailbox sind Beispiele für URIs. URLs (Uniform Resource Locator) sind übrigens eine Unterordnung von URIs, die Dateien referenzieren.

Listing 288 setzt einen `XmlReader` ein, um die Datei *Persons.xml* (Listing 281, Seite 425), die im Ordner der Anwendung erwartet wird, einzulesen und dabei gegen die Schema-Datei *Persons.xsd* (Listing 287) zu überprüfen. Zur Vereinfachung des Codes werden die eingelesenen Daten lediglich an der Konsole ausgegeben.

Um die Gültigkeit des Dokuments zu überprüfen, erfolgt das Einlesen in einer Ausnahmebehandlung. Damit später eine Aussage über die Gültigkeit getroffen werden kann, setze ich eine Variable `isValid` ein, die zunächst auf `true` und bei einer Ausnahme auf `false` gesetzt wird. Diese Variable wird am Ende des Einlesens abgefragt.

Das Beispiel erfordert den Import der Namensräume `System`, `System.IO`, `System.Reflection` und `System.Xml`.


```
// Der Dateiname der XML- und der Schema-Datei
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string xmlFileName = Path.Combine(appPath, "Persons.xml");
string xmlSchemaFileName = Path.Combine(appPath, "Persons.xsd");

// XmlReaderSettings erzeugen, den Validier-Typ auf "Schema"
// festlegen und das Schema ohne Ziel-Namensraum hinzufügen
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add(null, xmlSchemaFileName);

// XmlReader erzeugen
XmlReader xmlReader = XmlReader.Create(
   new FileStream(xmlFileName, FileMode.Open), settings);

// Merker für die Gültigkeit des Dokuments initialisieren
bool isValid = true;
try
{
   // Daten einlesen
   while (xmlReader.Read())
   {
      // Überprüfen, ob der aktuelle Knoten ein Element ist
      if (xmlReader.NodeType == XmlNodeType.Element)
      {
         // Überprüfen, um welches Element es sich handelt, und
         // die eingelesenen Daten entsprechend ausgeben
         if (xmlReader.Name == "person")
         {
            Console.WriteLine();
            Console.WriteLine("Person {0}", xmlReader.GetAttribute("id"));
         }
```

Listing 288: Validierendes Einlesen einer XML-Datei

```
         else if (xmlReader.Name == "firstname")
         {
            if (xmlReader.MoveToContent() != XmlNodeType.None)
            {
               Console.WriteLine("Vorname: {0}", xmlReader.ReadString());
            }
         }
         else if (xmlReader.Name == "lastname")
         {
            if (xmlReader.MoveToContent() != XmlNodeType.None)
            {
               Console.WriteLine("Nachname: {0}", xmlReader.ReadString());
            }
         }
         else if (xmlReader.Name == "type")
         {
            if (xmlReader.MoveToContent() != XmlNodeType.None)
            {
               Console.WriteLine("Typ: {0}", xmlReader.ReadString());
            }
         }
      }
   }
}
catch (Exception ex)
{
   // Fehler beim Lesen: Fehler ausgeben und Merker für die
   // Gültigkeit des Dokuments zurücksetzen
   Console.WriteLine(ex.Message);
   isValid = false;
}

// XmlValidatingReader schließen
xmlReader.Close();

// Ausgeben, ob die Datei gültig ist
Console.WriteLine();
if (isValid)
{
   Console.WriteLine("Die XML-Datei ist gültig");
}
else
{
   Console.WriteLine("Die XML-Datei ist nicht gültig");
}
```

Listing 288: Validierendes Einlesen einer XML-Datei (Forts.)

Abbildung 97 zeigt die Meldung des Programms für den Fall, dass das `id`-Attribut eines `person`-Elements einen nicht numerischen Wert speichert.



Abbildung 97: Fehlermeldung des Programms beim Einlesen einer Person.xml-Datei mit einem nicht numerischen id-Attribut in einem person-Element

174 XML-Dokumente auf Gültigkeit überprüfen

Basierend auf der im vorhergehenden Rezept verwendeten `XmlReader`-Klasse, die XML-Dateien gegen ein XML-Schema, ein DTD oder einfach nur auf Wohlgeformtheit überprüfen kann, habe ich eine Klasse `XmlValidator` mit den Methoden `ValidateXmlFile`, `ValidateXElement` und `ValidateXmlDocument` entwickelt, die eine XML-Datei, ein `XElement` (siehe Rezept 172) bzw. ein `XmlDocument`-Objekt daraufhin überprüfen, ob diese gültiges XML speichern. Diese Methoden sind folgendermaßen deklariert:

```
public XmlValidatorResult ValidateXmlFile(string xmlFileName,
   ValidationType validationType, string schemaUri)

public XmlValidatorResult ValidateXElement(XElement element,
   ValidationType validationType, string schemaUri)

public XmlValidatorResult ValidateXmlDocument(XmlDocument xmlDoc,
   ValidationType validationType, string schemaUri)
```

Am ersten Argument erwarten die Methoden den Dateinamen, ein `XElement` bzw. ein `XmlDocument`-Objekt. Am zweiten Argument definieren Sie den Validiertyp mit Werten der Aufzählung `ValidationType`. Die folgenden Konstanten sind verfügbar:

- `None`: Es wird keine Überprüfung gegen ein Schema oder DTD, sondern lediglich auf Wohlgeformtheit vorgenommen,
- `DTD`: Es wird gegen ein eingebettetes oder externes DTD geprüft,
- `Schema`: Es wird gegen ein eingebettetes oder externes Schema geprüft,
- `XDR`: Es wird gegen ein eingebettetes oder externes XDR-Schema geprüft.

Sie können explizit gegen ein externes Schema (kein DTD![3]) validieren, auf das im Dokument nicht verwiesen wird, indem Sie als Validiertyp `ValidationType.Schema` angeben und den Dateinamen der Schemadatei im letzten Argument übergeben.

Am dritten Argument übergeben Sie dann den URI einer externen Schemadatei, gegen die die Prüfung ausgeführt werden soll.

3. DTDs müssen über eine <!DOCTYPE-Deklaration in einem XML-Dokument angegeben werden, auch wenn es sich um ein externes DTD handelt. Es ist scheinbar nicht möglich, explizit gegen ein externes DTD zu validieren, das nicht im XML-Dokument deklariert ist.

Falls Sie gegen ein externes Schema validieren, und dieses wird über einen URI aus dem Internet gelesen (was problemlos möglich ist, sofern der Rechner eine Internetverbindung besitzt und der Zugriff auf das Internet für das Programm erlaubt ist), kann es sein, dass der Internetzugang über einen Proxy-Server erfolgt. In diesem Fall müssen Sie zumindest die Proxy-Adresse angeben. Erfordert der Proxy eine Authentifizierung, müssen Sie zudem die Benutzerdaten angeben. Die Klasse besitzt deswegen die Eigenschaften `ProxyAddress`, `ProxyUserName`, `ProxyUserPassword` und `ProxyUserDomain`. Wenn `ProxyAddress` ungleich `null` ist, werden diese Informationen verwendet.

`ValidateXmlFile`, `ValidateXElement` und `ValidateXmlDocument` geben einen Wert der (eigenen) Aufzählung `XmlValidatorResult` zurück:

- `XmlValidatorResult.Valid`: Das Dokument ist in Ordnung,
- `XmlValidatorResult.ErrorsExists`: Das Dokument enthält Fehler,
- `XmlValidatorResult.WarningsExists`: Das Dokument ist prinzipiell in Ordnung, es wurden jedoch Warnungen generiert.

Falls Warnungen generiert wurden oder Fehler aufgetreten sind, können Sie diese aus den Eigenschaften `Warnings` bzw. `Errors` auslesen, die beide einfache `List`-Auflistungen vom Typ `string` sind. Damit diese Auflistungen zur Verfügung stehen, werden sie im Konstruktor der `XmlValidator`-Klasse erzeugt.

Zum Kompilieren dieser Klasse müssen Sie die Namensräume `System`, `System.IO`, `System.Xml`, `System.Xml.Schema` und `System.Collections.Generic` importieren.

Die Klasse beginnt mit der Deklaration der Eigenschaften.

```
public class XmlValidator
{
   private List<string> errors;
   /* Verwaltet alle beim Validieren aufgetretenen Fehler */
   public List<string> Errors
   {
      get
      {
         return this.errors;
      }
   }

   private List<string> warnings;
   /* Verwaltet alle beim Validieren aufgetretenen Warnungen */
   public List<string> Warnings
   {
      get
      {
         return this.warnings;
```

Listing 289: Die Eigenschaften der XmlValidator-Klasse

```
    }
}

/* Proxy- Informationen*/
public string ProxyAddress = null;
public string ProxyUserName = null;
public string ProxyUserPassword = null;
public string ProxyUserDomain = null;
```

Listing 289: Die Eigenschaften der XmlValidator-Klasse (Forts.)

Dann folgen der Rückgabetyp der `ValidateXml`-Methoden und der Konstruktor, der die Warnungs- und die Fehler-Auflistung erzeugt:

```
public enum XmlValidatorResult
{
   Valid,         /* Das Dokument ist gültig */
   ErrorsExists,  /* Es sind Fehler aufgetreten */
   WarningsExists /* Es sind Warnungen gemeldet worden */
}

/* Konstruktor  */
public XmlValidator()
{
   this.errors = new List<string>();
   this.warnings = new List<string>();
}
```

Listing 290: Der Rückgabetyp der ValidateXml-Methoden und Konstruktor

Wenn Sie an die `ValidationEventHandler`-Eigenschaft des `XmlReaderSettings`-Objekt eine passende Methode hängen, wird beim Auftreten eines Fehlers oder einer Warnung während der Validierung keine Ausnahme generiert, sondern diese Methode aufgerufen. Die `ValidationCallBack`-Methode der `XmlValidator`-Klasse nutzt diese Tatsache um die aufgetretenen Warnungen und Fehler an die `Warnings`- bzw. `Errors`-Auflistung zu hängen:

```
private void ValidationCallBack(object sender, ValidationEventArgs args)
{
   // Fehler bzw. Warnung hinzufügen
   if (args.Severity == XmlSeverityType.Error)
   {
      this.Errors.Add(args.Message);
   }
   else
   {
```

Listing 291: Behandlungsmethode für das Validation-Ereignis

```
            this.Warnings.Add(args.Message);
        }
    }
```

Listing 291: Behandlungsmethode für das Validation-Ereignis (Forts.)

Falls das Schema-Dokument aus dem Internet geladen wird und der Internetzugang über einen Proxy-Server erfolgt, müssen die Proxy-Informationen ausgewertet werden. Leider scheint es keine Möglichkeit zu geben, dem `XmlReader` oder der `XmlReaderSettings`-Instanz die notwendigen Proxy-Informationen direkt zu übergeben. Dieses Problem können Sie nach meinen Recherchen nur über einen *Resolver*[4] lösen. In unserem Fall handelte sich dabei um eine Instanz einer von der Klasse `XmlUrlResolver` abgeleiteten Klasse. Diese hängen Sie an die `XmlResolver`-Eigenschaft des `XmlReaderSettings`-Objekts. Der `XmlReader` ruft die Methode `GetEntity` für jedes Dokument auf, das nicht direkt geladen werden kann. Das kann einmal das Schema-Dokument selbst sein, aber auch weitere Dokumente, auf die im Schema-Dokument verwiesen wird[5]. Der Resolver-Methode wird u. a. der URI des zu ladenden Dokuments übergeben. Die Methode muss das Dokument laden und als Stream zurückgeben.

Für unseren Fall habe ich dazu eine Klasse `XmlUrlProxyResolver` implementiert. Neben der genannten Methode besitzt diese Klasse die Eigenschaft `Proxy`, die ein Objekt vom Typ `IWebProxy` verwaltet. Dieses Objekt wird später mit den Proxy-Informationen initialisiert. Die Methode `GetEntity` erzeugt zur Auflösung des Zugriffs-Problems eine Instanz der `WebRequest`-Klasse, initialisiert diese mit der URI des abzurufenden Dokuments und dem Proxy-Objekt, liest den `Response`-Stream ein und gibt diesen zurück.

```
private class XmlUrlProxyResolver : XmlUrlResolver
{
    /* Verwaltet die Proxy-Informationen */
    public IWebProxy Proxy = new WebProxy();

    /* Liefert einen Stream, der die Daten der über das
       Argument absoluteUri definierten Datei enthält */
    public override object GetEntity(Uri absoluteUri, string role,
        Type ofObjectToReturn)
    {
        // Neuen Request erzeugen
        WebRequest request = WebRequest.Create(absoluteUri);

        // Proxy übergeben
        request.Proxy = this.Proxy;
```

Listing 292: Klasse zur Auflösung von Schema-Lade-Problemen wegen fehlender Proxy-Informationen

4. Resolver = Auflöser
5. In der Praxis werden Schemata häufig in mehrere Dokumente aufgeteilt, die in das Haupt-Schema-Dokument eingebunden werden.

```
        // Response und Response-Stream holen und zurückgeben
        WebResponse response = request.GetResponse();
        Stream stream = response.GetResponseStream();
        return stream;
    }
}
```

Listing 292: Klasse zur Auflösung von Schema-Lade-Problemen wegen fehlender Proxy-Informationen (Forts.)

Die eigentliche Validierung übernimmt die private Methode `ValidateXml`, die ein `XmlReader`-Objekt und die Validierinformationen übergeben bekommt. `ValidateXml` erzeugt zunächst eine `XmlReaderSettings`-Instanz für die Validier-Einstellungen. Nachdem die Validierinformationen in dieses Objekt geschrieben wurden, erzeugt `ValidateXml` eine neue Instanz der privaten `XmlUrlProxyResolver`-Klasse, initialisiert diese mit den Proxy-Informationen (falls eine Proxy-Adresse angegeben wurde) und übergibt diese an die Eigenschaft `XmlResolver` der `XmlReaderSettings`-Instanz. Nachdem die `ValidationCallBack`-Methode an die Eigenschaft `ValidationEventHandler` gehängt wurde, erzeugt `ValidateXml` einen neuen `XmlReader` und liest die Daten sequenziell ein um diese zu überprüfen. Nach dem Einlesen werden die `Warnings`- und die `Errors`-Auflistung überprüft, um zu entscheiden, welches Ergebnis zurückgegeben werden soll.

```
private XmlValidatorResult ValidateXml(XmlReader xmlReader,
   ValidationType validationType, string schemaUri)
{
   // Eventuelle alte Fehler und Warnungen löschen
   this.errors.Clear();
   this.warnings.Clear();

   // XmlReaderSettings erzeugen und den Validier-Typ festlegen
   XmlReaderSettings settings = new XmlReaderSettings();
   settings.ValidationType = validationType;

   // Das externe Schema hinzufügen, wenn ein Dateiname angegeben wurde
   if (schemaUri != null)
   {
      try
      {
         settings.Schemas.Add(null, schemaUri);
      }
      catch (Exception ex)
      {
         throw new XmlException("Fehler beim Hinzufügen des externen " +
            "Schemas ': " + schemaUri + "' " + ex.Message, ex);
      }
```

Listing 293: Die private Methode, die die Validierung vornimmt

```
}
// Proxy-Informationen über die eigene Resolver-Klasse übergeben
if (this.ProxyAddress != null)
{
   XmlUrlProxyResolver resolver = new XmlUrlProxyResolver();
   ICredentials credentials = new NetworkCredential(this.ProxyUserName,
      this.ProxyUserPassword, this.ProxyUserDomain);
   resolver.Proxy = new WebProxy(this.ProxyAddress, true, new
      string[] { }, credentials);
   settings.XmlResolver = resolver;
}

// Delegat mit der Methode für das Validation-Ereignis übergeben
settings.ValidationEventHandler += this.ValidationCallBack;

// XmlReader mit den Einstellungen erzeugen
XmlReader validationReader = null;
try
{
   validationReader = XmlReader.Create(xmlReader, settings);
}
catch (XmlException ex)
{
   throw new XmlException("Fehler beim Einlesen des XML-Dokuments: " +
      ex.Message, ex);
}

// Daten einlesen und validieren
try
{
   while (validationReader.Read())
   {
   }
}
catch (Exception ex)
{
   this.Errors.Add(ex.Message);
}

// Ergebnis zurückgeben
if (this.Errors.Count == 0 && this.Warnings.Count == 0)
{
   return XmlValidatorResult.Valid;
}
else if (this.Errors.Count == 0)
```

Listing 293: Die private Methode, die die Validierung vornimmt (Forts.)

```
      {
         return XmlValidatorResult.WarningsExists;
      }
      else
      {
         return XmlValidatorResult.ErrorsExists;
      }
   }
```

Listing 293: Die private Methode, die die Validierung vornimmt (Forts.)

Für den Fall, dass ein externes DTD oder Schema oder die XML-Datei nicht gelesen werden kann, erzeugt `ValidateXml` Ausnahmen vom Typ `XmlException`. Damit ist sichergestellt, dass die Validierung nur dann ausgeführt wird, wenn alle verbundenen Dokumente vorhanden bzw. bei den DTD- und Schema-Dokumenten auch gültig sind.

Die öffentliche Methode `ValidateXmlFile` erzeugt nun einfach ein `XmlReader`-Objekt über den übergebenen Dateinamen und ruft damit `ValidateXml` auf.

```
public XmlValidatorResult ValidateXmlFile(string xmlFileName,
   ValidationType validationType, string schemaUri)
{
   // XmlReader erzeugen und damit ValidateXml aufrufen
   return ValidateXml(XmlReader.Create(xmlFileName),
      validationType, schemaUri);
}
```

Listing 294: Öffentliche Methode zum Validieren einer XML-Datei

Die Methode ValidateXElement validiert ein XElement:

```
public XmlValidatorResult ValidateXElement(XElement element,
   ValidationType validationType, string schemaUri)
{
   return ValidateXml(element.CreateReader(), validationType, schemaUri);
}
```

Listing 295: Methode zum Validieren eines in einem XElement verwalteten XML-Dokuments

Die ebenfalls öffentliche Methode `ValidateXmlDocument`, mit der die Klasse `XmlValidator` endet, ist ein wenig komplexer (und etwas »tricky« ☺). Um aus dem übergebenen `ValidateXmlDocument`-Objekt ein `XmlReader`-Objekt zu erhalten, erzeugt `ValidateXmlDocument` zunächst mit Hilfe der `Save`-Methode des `XmlDocument`-Objekts einen `MemoryStream`. Dieser Stream wird dann als Basis für das `XmlReader`-Objekt verwendet. Dabei ist es wichtig, dass der Stream jeweils auf die Anfangsposition gesetzt wird, da die aktuelle Position nach dem Schreiben und nach dem Erzeugen des `XmlReader`-Objekts auf dem Ende des Stream steht und das `XmlReader`-Objekt in diesem Fall die Daten nicht lesen könnte (was zu einer Ausnahme mit der nicht gerade aussa-

gefähigen Meldung »Systemfehler« führen würde). `ValidateXmlDocument` ruft deshalb jeweils die `Seek`-Methode des Stream auf um diesen an die Anfangsposition zu setzen. Mit dem erzeugten `XmlReader`-Objekt ruft `ValidateXmlDocument` dann nur noch `ValidateXml` auf.

```
   public XmlValidatorResult ValidateXmlDocument(XmlDocument xmlDoc,
      ValidationType validationType, string schemaUri)
   {
      // MemoryStream aus dem Xml-Dokument erzeugen
      MemoryStream xmlStream = new MemoryStream();
      xmlDoc.Save(xmlStream);
      xmlStream.Seek(0, SeekOrigin.Begin);

      // Mit dem MemoryStream eine XmlReader-Instanz erzeugen
      XmlReader XmlReader = XmlReader.Create(xmlStream);
      xmlStream.Seek(0, SeekOrigin.Begin);

      return ValidateXml(XmlReader, validationType, schemaUri);
   }
}
```

Listing 296: Öffentliche Methode zum Validieren eines XML-Dokuments und Ende der XmlValidator-Klasse

Das folgende Listing zeigt eine beispielhafte Anwendung der Klasse `XmlValidator`, bei der die XML-Datei *Person.xml* gegen ein externes Schema *Person.xsd* geprüft wird. Das Schema wird von einer Internetadresse geladen. Proxy-Informationen werden hier nicht angegeben, sind aber für den Fall, dass Sie diese angeben müssen, auskommentiert.

Das Programm benötigt die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Xml`, `System.Collections` und `System.Windows.Forms`.

```
string xmlFilename = Path.Combine(Application.StartupPath, "Persons.xml");
string schemaUri = "http://www.juergen-bayer.net/buecher/csharpcodebook3" +
   "/daten/Persons.xsd";
//string schemaUri = Path.Combine(Application.StartupPath, "Persons.xsd");

// XmlValidator erzeugen
XmlValidator xmlValidator = new XmlValidator();
XmlValidator.XmlValidatorResult result;

// Proxy-Informationen übergeben
//xmlValidator.ProxyAddress = "10.0.0.1";
```

Listing 297: Prüfen einer XML-Datei gegen ein externes Schema über die Klasse XmlValidator

```
//xmlValidator.ProxyUserName = "user";
//xmlValidator.ProxyUserPassword = "password";

try
{
   result = xmlValidator.ValidateXmlFile(xmlFilename,
      ValidationType.Schema, schemaUri);
}
catch (XmlException ex)
{
   Console.WriteLine(ex.Message);
   return;
}

switch (result)
{
   case XmlValidator.XmlValidatorResult.Valid:
      Console.WriteLine("Das XML-Dokument ist valide.");
      break;

   case XmlValidator.XmlValidatorResult.WarningsExists:
      Console.WriteLine("Das XML-Dokument ist valide, " +
         "es wurden aber Warnungen gemeldet:");
      for (int i = 0; i < xmlValidator.Warnings.Count; i++)
      {
         Console.WriteLine("\r\n* " + xmlValidator.Warnings[i].ToString());
      }
      break;

   case XmlValidator.XmlValidatorResult.ErrorsExists:
      Console.WriteLine("Das XML-Dokument ist nicht valide.");
      // Fehler ausgeben
      Console.WriteLine("Die folgenden Fehler wurden gefunden:");
      for (int i = 0; i < xmlValidator.Errors.Count; i++)
      {
         Console.WriteLine("\r\n* " + xmlValidator.Errors[i].ToString());
      }
      // Überprüfen, ob Warnungen vorhanden sind, und diese ebenfalls
      // ausgeben
      if (xmlValidator.Warnings.Count > 0)
      {
         Console.WriteLine("Die folgenden Warnungen wurden gemeldet:");
         for (int i = 0; i < xmlValidator.Warnings.Count; i++)
```

Listing 297: Prüfen einer XML-Datei gegen ein externes Schema über die Klasse XmlValidator (Forts.)

```
         {
            Console.WriteLine("\r\n* " +
               xmlValidator.Warnings[i].ToString());
         }
      }
      break;
}
```

Listing 297: Prüfen einer XML-Datei gegen ein externes Schema über die Klasse XmlValidator (Forts.)

Wird die XML-Datei in Listing 298 (die einige Fehler enthält) gegen das Schema in Listing 287 (Seite 437) geprüft, resultiert die Ausgabe des Programms in Abbildung 98.

```
<?xml version="1.0" encoding="utf-8"?>
<persons>
   <person id="1000">
      <firstname>Zaphod</firstname>
      <lastname>Beeblebrox</lastname>
      <!-- type fehlt -->
   </person>
   <person id="abc"> <!-- id ist nicht numerisch -->
      <!-- firstname fehlt -->
      <lastname>Prefect</lastname>
      <type>Alien</type>
   </person>
   <person id="1002">
      <firstname>Tricia</firstname>
      <!-- lastname fehlt -->
      <type>Earthling</type>
   </person>
   <person id="1003">
      <firstname>Arthur</firstname>
      <lastname>Dent</lastname>
      <type>Earthling</type>
   </person>
</persons>
```

Listing 298: Eine ungültige XML-Datei

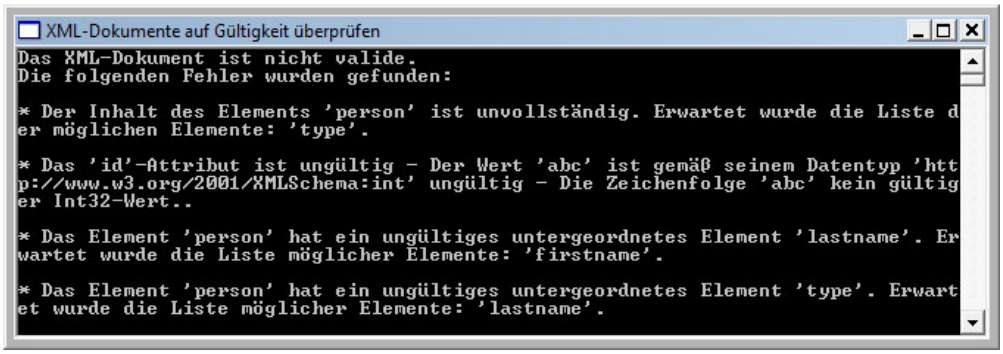


Abbildung 98: Das Beispielprogramm meldet Fehler in der XML-Datei

175 XML-Dokumente validierend über das X-DOM lesen

In Rezept 172 wurden die XML-Daten über ein `XElement`-Objekt so eingelesen, dass keine Validierung gegenüber einem eventuell vorhandenen DTD oder Schema erfolgt. Um dies zu erreichen, müssen Sie der `Load`-Methode des `XElement`- oder `XDocument`-Objekts ein `XmlValidatingReader`-Objekt übergeben, das Sie zuvor entsprechend der gewünschten Validierung initialisieren (vergleichen Sie hierzu das Rezept 174). `Load` erzeugt eine Ausnahme vom Typ `XmlException`, wenn die eingelesenen Daten nicht dem Schema bzw. DTD entsprechen.

Das folgende Listing liest die Datei *Persons.xml* (Listing 281 in Rezept 171) über ein `XDocument`-Objekt ein. Über eine `XmlValidatingReader`-Instanz wird das Dokument dabei gegen das externe Schema *Persons.xsd* (Listing 287 in Rezept 173) validiert. Aus Vereinfachungsgründen werden die eingelesenen Daten lediglich an der Konsole ausgegeben.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Xml` und `System.Xml.Linq` importieren.

```
// Der Dateiname der XML-Datei und der Schema-Datei
string xmlFileName = "Persons.xml";
string xsdFileName = "Persons.xsd";

// Die XML-Datei in ein XElement laden
XElement rootElement = null;

// XML-Datei laden
try
{
   // XmlReader zum validierenden Lesen erzeugen und initialisieren
   XmlReaderSettings settings = new XmlReaderSettings();
   settings.ValidationType = ValidationType.Schema;
   settings.Schemas.Add(null, xsdFileName);
   XmlReader xmlReader = XmlReader.Create(xmlFileName, settings);
```

Listing 299: Validierendes Einlesen einer XML-Datei über das DOM

```
   // XmlReader zum Laden des Dokuments verwenden
   rootElement = XElement.Load(xmlReader);
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Einlesen der XML-Datei: " +
      ex.Message);
   Console.ReadLine();
   return;
}

// Alle person-Elemente einlesen und durchgehen
var personElements = rootElement.Elements("person");
foreach (var personElement in personElements)
{
   Console.WriteLine("\r\nPerson");

   // Das Attribut id einlesen
   XAttribute idAttribute = personElement.Attribute("id");
   if (idAttribute != null)
   {
      Console.WriteLine("Id: {0}", idAttribute.Value);
   }

   // firstname-Element suchen
   XElement firstNameElement = personElement.Element("firstname");
   if (firstNameElement != null)
   {
      Console.WriteLine("Vorname: {0}", firstNameElement.Value);
   }

   // lastname-Element suchen
   XElement lastNameElement = personElement.Element("lastname");
   if (lastNameElement != null)
   {
      Console.WriteLine("Nachname: {0} ", lastNameElement.Value);
   }
}
```

Listing 299: Validierendes Einlesen einer XML-Datei über das DOM (Forts.)

Hinweis

In diesem Beispiel wird das Programm abgebrochen, sobald nur ein Fehler auftritt. Sie können jedoch auch über die Zuweisung einer Ereignisbehandlungsmethode an die Eigenschaft `ValidationEventHandler` des `XmlReaderSettings`-Objekts erreichen, dass keine Ausnahme generiert, sondern für jeden Fehler diese Methode aufgerufen wird. Damit können Sie (mit ein wenig Programmierung ☺) auch alle Fehler auswerten. Vergleichen Sie dazu das Rezept 174.

Abbildung 99 zeigt die Meldung des Programms für den Fall, dass in einem `person`-Element im Attribut `id` ein nicht numerischer Wert gespeichert ist. In der Schema-Datei (siehe Rezept 173) ist dieses Attribut als `int`-Wert definiert.



Abbildung 99: Fehlermeldung des Beispielprogramms beim Einlesen einer ungültigen XML-Datei

176 Gezielt über XPath auf Elemente in einem XML-Dokument ohne Namensraum zugreifen

Das X-DOM (siehe Rezept 172) ermöglicht in Form der `XElement`- und der `XDocument`-Klasse einen sehr flexiblen Zugriff auf XML-Dokumente über das X-DOM-Objektmodell oder über LINQ. In manchen Fällen ist aber auch der Zugriff über XPath sinnvoll. Ein Vorteil ist z. B., dass in einem XPath-Ausdruck ein komplexer Pfad in einem XML-Dokument abgebildet werden kann.

XPath ist vom W3C als Sprache zur Adressierung von Teilen eines XML-Dokuments vorgesehen. XPath ist recht komplex und erlaubt eine sehr flexible Adressierung in einem XML-Dokument. Die Empfehlung des W3C finden Sie an der Adresse *www.w3.org/TR/xpath*. Eine sehr gute Einführung zu XPath finden Sie an der Adresse *www.w3schools.com/xpath/default.asp*.

Um auf ein Element in einem XML-Dokument ohne Zuordnung der Elemente zu einem Namensraum zuzugreifen, können Sie einen einfachen XPath-Ausdruck verwenden. Ein solcher setzt sich zusammen aus hierarchisch geordneten Element-Namen, die jeweils durch einen Schrägstrich voneinander getrennt werden. Ein Schrägstrich am Anfang bezeichnet das Wurzelelement. Um auf alle `person`-Elemente der XML-Datei aus dem Rezept 171 zuzugreifen, würde der XPath-Ausdruck folgendermaßen aussehen:

```
/persons/person
```

Zum Einlesen aller `firstname`-Knoten würden Sie den folgenden XPath-Ausdruck verwenden:

```
/persons/person/firstname
```

Um über das X-DOM mit XPath arbeiten zu können, müssen Sie den Namensraum `System.Xml.XPath` einbinden. Dieser Namensraum enthält die Klasse `Extensions` mit Erweiterungsmethoden für die Arbeit mit XPath.

Die Methode `XPathSelectElement` liefert ein `XElement`, das dem übergebenen XPath-Ausdruck entspricht. `XPathSelectElements` liefert eine Auflistung von Elementen.

Das Einlesen der Vornamen aller im Dokument gespeicherten Personen würde also z. B. so aussehen:

```
using System;
using System.Xml.Linq;
using System.Xml.XPath;

...

// Der Dateiname
string xmlFileName = "Persons.xml";

// XElement-Instanz erzeugen
XElement rootElement = XElement.Load(xmlFileName);

// Suchen aller XML-Unterknoten mit einem bestimmten Namen
string xPath = "/persons/person/firstname";
var firstNameElements = rootElement.XPathSelectElements(xPath);
if (firstNameElements != null)
{
   foreach (var firstNameElement in firstNameElements)
   {
      Console.WriteLine(firstNameElement.Value);
   }
}
```

Listing 300: Suchen aller XML-Unterelemente mit einem bestimmten Namen

Wenn Sie gezielt auf ein Element mit einem bestimmten Attributwert zugreifen wollen, erweitern Sie den XPath-Ausdruck um einen *Knotentest* mit der Angabe des Attributnamens und des Werts des Attributs in der folgenden Form:

```
[attribute::Name='Wert']
```

Zum Einlesen der Person mit der ID 1001 verwenden Sie dann z. B. den folgenden Programmcode:

```
// Suchen eines XML-Knotens über einen Attributwert
// Die Person mit der Id 1001 suchen
xPath = "/persons/person[attribute::id='1001']";
XElement personElement = rootElement.XPathSelectElement(xPath);
if (personElement != null)
{
   // Element für die Person mit der Id 1001 gefunden
   ...
}
```

Listing 301: Suchen eines XML-Knotens über einen Attributwert

Suchen Sie nach Knoten, deren Unterknoten einen bestimmten Wert speichern, erweitern Sie den XPath-Ausdruck um einen Knotentest in der folgenden Form:

```
[Name='Wert']
```

Wenn Sie z. B. alle Personen einlesen wollen, deren Typ *Alien* ist, können Sie den folgenden Programmcode verwenden:

```
// Suchen mehrerer XML-Knoten über den Wert des Inhalts eines Unterknotens:
// Personen mit Typ = Alien suchen
var alienElements = rootElement.XPathSelectElements("/persons/person[type='Alien']");
if (alienElements != null)
{
   // Passende person-Knoten wurden gefunden:
   // Alle gefundenen Personen durchgehen
   foreach (var alienElement in alienElements)
   {
      // Element verarbeiten
   }
}
```

Listing 302: Suchen mehrerer XML-Elemente über den Wert des Inhalts eines Unterelements

Hinweis

LINQ to XML bietet mit seinen Fähigkeiten ähnliche Features. Das Beispiel könnte problemlos auch über LINQ to XML mit entsprechenden LINQ-Abfragen umgesetzt werden. XPath könnte aber in der Praxis den Vorteil haben, dass ein XPath-Ausdruck auch zwischen verschiedenen Parteien, die mit XML-Dokumenten arbeiten, ausgetauscht werden kann (was bei LINQ to XML nicht möglich ist).

177 Gezielt über XPath auf Elemente in einem XML-Dokument mit Namensraum zugreifen

Etwas komplizierter als der XPath-Zugriff auf XML-Elemente in Dokumenten ohne Namensraum (siehe Rezept 176) wird die Suche, wenn die gesuchten Elemente innerhalb des XML-Dokuments einem XML-Namensraum zugeordnet sind. XML-Namensräume werden in Rezept 170 beschrieben.

Um in einem XML-Dokument mit Namensraum-Zuordnung über `XPath` zu suchen, müssen Sie die alte Namensraum-Auflöse-Technik von .NET verwenden. Dazu müssen Sie das XML-Dokument über einen `XmlReader` lesen. Dann erzeugen Sie ein `XmlNamespaceManager`-Objekt, dem Sie am Konstruktor das `NameTable`-Objekt des `XmlReader` übergeben. Über die `AddNamespace`-Methode fügen Sie dann einen Namensraum hinzu. Am ersten Argument dieser Methode übergeben Sie ein Kürzel, das Sie in den XPath-Ausdrücken für den Namensraum verwenden wollen. Am zweiten Argument übergeben Sie den Namensraum (als String).

Da Sie beim Zugriff über die X-DOM-Methoden (`Elements`, `Element` etc.) zudem (idealerweise) ein `XNamespace`-Objekt benötigen, sollten Sie dieses zuvor erzeugen und dessen Namensraum-String an `AddNamespace` anfügen.

Wenn Sie dann über `XPathSelectElement` oder `XPathSelectElements` suchen, übergeben Sie am zweiten Argument das `XmlNamespaceManager`-Objekt. Der XPath-Ausdruck muss dann nur noch um den im `XmlNamespaceManager` definierten Namensraum-Präfix erweitert werden. Diesen Präfix müssen Sie vor jeden Namen eines Elementes schreiben, das einem Namensraum zugeordnet ist (aber nicht vor Attributen, die nicht explizit einem Namensraum zugeordnet sind).

Das Beispiel in Listing 304 zeigt eine XPath-Abfrage in einem XML-Dokument mit Namensraum in Form der Suche nach Personen mit dem Wert »Alien« im *type*-Element. Die Elemente des XML-Dokuments sind dabei dem Namensraum *http://www.addison-wesley.de/codebook* zugeordnet (Listing 304).


```
<?xml version="1.0" encoding="utf-8" ?>
<persons xmlns ="http://www.addison-wesley.de/codebook">
  <person id="1000">
    <firstname>Zaphod</firstname>
    <lastname>Beeblebrox</lastname>
    <type>Alien</type>
  </person>

  <person id="1001">
    <firstname>Ford</firstname>
    <lastname>Prefect</lastname>
    <type>Alien</type>
  </person>
</persons>
```

Listing 303: Das (verkürzte) XML-Dokument mit Namensraumzuordnung

Das Beispiel erfordert den Import der Namensräume `System`, `System.Xml`, `System.Xml.Linq` und `System.Xml.XPath`.

```
// XElement-Instanz über einen XmlReader erzeugen
XmlReader xmlReader = XmlReader.Create(xmlFileName);
XElement rootElement = XElement.Load(xmlReader);

// XmlNamespaceManager erzeugen
XmlNamespaceManager namespaceManager =
   new XmlNamespaceManager(xmlReader.NameTable);
XNamespace ns = "http://www.addison-wesley.de/codebook";
namespaceManager.AddNamespace("aw", ns.NamespaceName);

// Suchen mehrerer XML-Knoten über den Wert des Inhalts eines Unterknotens
var alienElements = rootElement.XPathSelectElements(
  "aw:person[aw:type='Alien']", namespaceManager);
```

Listing 304: XPath-Suche in einem XML-Dokument mit Namensraum-Zuordnung

```
if (alienElements != null)
{
   // Passende person-Knoten wurden gefunden:
   // Alle gefundenen Personen durchgehen
   foreach (var alienElement in alienElements)
   {
      // Das Attribut id einlesen
      XAttribute idAttribute = alienElement.Attribute("id");
      if (idAttribute != null)
      {
         Console.WriteLine(idAttribute.Value);
      }

      // firstname-Element suchen
      XElement firstNameElement = alienElement.Element(ns + "firstname");
      if (firstNameElement != null)
      {
         Console.WriteLine(firstNameElement.Value);
      }

      // lastname-Element suchen
      XElement lastNameElement = alienElement.Element(ns + "lastname");
      if (lastNameElement != null)
      {
         Console.WriteLine(lastNameElement.Value);
      }
      Console.WriteLine();
   }
}
else
{
   Console.WriteLine("Keine Aliens gefunden.");
}
```

Listing 304: XPath-Suche in einem XML-Dokument mit Namensraum-Zuordnung (Forts.)

178 XML-Dokumente mit einem XmlWriter erzeugen

Die Verwendung eines `XmlWriter`-Objekts ist die effizienteste Methode, XML-Dokumente komplett neu zu erstellen. Über ein solches Objekt erzeugen Sie das XML-Dokument sequenziell, im Gegensatz zum Schreiben über ein `XElement`- oder `XDocument`-Objekt, das ein beliebiges Einfügen von Knoten erlaubt.

Einen `XmlWriter` können Sie über die statische `Create`-Methode erzeugen. Dieser Methode können Sie neben einem vorhandenen `XmlTextWriter`-Objekt ein `Stream`-Objekt, das die zu erzeugende Datei repräsentiert, oder den Namen der zu erzeugenden Datei übergeben.

Am zweiten Argument können Sie eine `XmlWriterSettings`-Instanz mit Einstellungen übergeben. Über die Eigenschaften `Indent` und `IndentChars` legen Sie fest, ob und mit welchen Zeichen eingerückt wird. Die Eigenschaft `Encoding` legt die Codierung fest. Voreingestellt ist UTF-8.

Dann schreiben Sie die einzelnen Bestandteile des Dokuments über eine Vielzahl an *Write*-Methoden (von denen ich hier nur die wichtigsten beschreibe).

`WriteStartDocument` erzeugt zunächst den Beginn des Dokuments. Über das optionale boolesche Argument können Sie festlegen, welchen Wert das `standalone`-Attribut besitzen soll.

Einzelne Elemente ohne Textdaten erzeugen Sie über `WriteStartElement`. Dabei können Sie (u. a.) den Namen des Elements und optional den Namensraum[6] übergeben, dem das Element zugeordnet sein soll. Attribute eines Elements können Sie über `WriteAttributeString` schreiben.

Elemente mit Textdaten erzeugen Sie auf eine einfache Weise über die Methode `WriteElementString`. Diese Methode erzeugt das Startelement, den Textinhalt und direkt auch das Endelement. Alternativ können Sie auch zunächst `WriteStartElement` aufrufen, dann den Inhalt über `WriteString` oder über eine andere Methode wie `WriteBase64` (binäre Daten Base64-codiert schreiben) oder `WriteChars` (Byte-Array als String schreiben) erzeugen und das Element schließlich über `WriteEndElement` abschließen. `WriteEndElement` schließt immer das Element ab, das zuletzt über `WriteStartElement` begonnen wurde. Auf diese Weise müssen Sie alle »offenen« Elemente abschließen.

Am Ende der Datei rufen Sie dann noch `WriteEndDocument` auf um das Dokument abzuschließen.

Das Programm in Listing 306 erzeugt eine einfache XML-Datei mit den Daten zweier Personen. Diese Datei sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<persons xmlns="http://www.addison-wesley.de/codebook/csharp">
   <!--Beispiel zum Erzeugen von XML-Dateien-->
   <person id="1001">
      <firstname>Zaphod</firstname>
      <lastname>Beeblebrox</lastname>
      <type>Alien</type>
   </person>
   <person id="1002">
      <firstname>Tricia</firstname>
      <lastname>McMillan</lastname>
      <type>Earthling</type>
   </person>
</persons>
```

Listing 305: Die vom Beispiel erzeugte XML-Datei

6. XML-Namensräume, die üblicherweise (nicht unbedingt existenten) Internet-Adressen entsprechen, sichern ab, dass Elemente (über den Namensraum) auch dann eindeutig identifiziert werden können, wenn mehrere XML-Dokumente zu einem Dokument zusammengeführt werden. Wenn Sie ein Element einem Namensraum zuordnen, sind automatisch auch alle untergeordneten Elemente diesem Namensraum zugeordnet.

Das Beispiel erfordert den Import der Namensräume `System`, `System.Diagnostics`, `System.IO`, `System.Reflection` und `System.Xml`.

```
// Der Dateiname der XML-Datei
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string xmlFileName = Path.Combine(appPath, "persons.xml");

// XmlWriter für eine UTF-8-Codierung mit Einrückung erzeugen
XmlWriter xmlWriter = null;
try
{
   XmlWriterSettings settings = new XmlWriterSettings
   {
      Indent = true,
      IndentChars = "   "
   };
   xmlWriter = XmlWriter.Create(xmlFileName, settings);
}
catch (Exception ex)
{
   Console.WriteLine("Fehler beim Öffnen der XML-Datei: " + ex.Message);
   return;
}

// Dokument-Beginn mit standalone-Attribut schreiben
xmlWriter.WriteStartDocument(true);

// Startelement mit Namensraum schreiben
xmlWriter.WriteStartElement("persons",
   "http://www.addison-wesley.de/codebook/csharp");

// Kommentar schreiben
xmlWriter.WriteComment("Beispiel zum Erzeugen von XML-Dateien");

// Element für eine Person erzeugen
xmlWriter.WriteStartElement("person");

// Attribut id schreiben
xmlWriter.WriteAttributeString("id", "1001");

// Unterelemente mit String-Daten erzeugen
xmlWriter.WriteElementString("firstname", "Zaphod");
xmlWriter.WriteElementString("lastname", "Beeblebrox");
xmlWriter.WriteElementString("type", "Alien");
```

Listing 306: Schreiben einer XML-Datei über ein XmlWriter-Objekt

```
// person-Element abschließen
xmlWriter.WriteEndElement();

// Element für eine weitere Person erzeugen
xmlWriter.WriteStartElement("person");

// Attribut id schreiben
xmlWriter.WriteAttributeString("id", "1002");

// Unterelemente mit String-Daten erzeugen
xmlWriter.WriteElementString("firstname", "Tricia");
xmlWriter.WriteElementString("lastname", "McMillan");
xmlWriter.WriteElementString("type", "Earthling");

// person-Element abschließen
xmlWriter.WriteEndElement();

// Dokument abschließen
xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();

// XmlWriter schließen
xmlWriter.Close();
```

Listing 306: Schreiben einer XML-Datei über ein XmlWriter-Objekt (Forts.)

179 XML-Dokumente über das X-DOM erzeugen und ändern

Wie ich bereits im Rezept 172 beschrieben habe, implementieren die zu LINQ to XML gehörenden Klassen (`XElement`, `XDocument` etc.) eine Microsoft-Abwandlung des W3C-DOM (Document Object Model). Das von mir und anderen Autoren als *X-DOM* bezeichnete LINQ-to-SQL-DOM repräsentiert ein XML-Dokument im Arbeitsspeicher in Form von hierarchisch angeordneten Objekten. Zum Erstellen eines XML-Dokuments haben Sie nun zwei Möglichkeiten: die DOM-ähnliche Methode und die so genannte funktionale Konstruktion.

Die DOM-ähnliche Methode

`XContainer`, die Basisklasse aller Knoten, die untergeordnete Knoten besitzen können, erlaubt es, über die `Add`-Methode einem XML-Container beliebige Objekte unterzuordnen. Der Typ des Arguments dieser Methode ist `Object`. Sie können also prinzipiell jedes Objekt übergeben. Objekte, die nicht von `XObject` (der Basisklasse aller X-DOM-Klassen) abgeleitet sind, werden allerdings (über `ToString`) in einen String umgewandelt. Ansonsten können Sie auch Instanzen der X-DOM-Klassen übergeben, um den Baum des XML-Dokuments aufzubauen. Das Ganze ist – verglichen mit dem alten DOM über `XmlDocument`, `XmlElement` etc. – sehr einfach.

Ich baue auf diese Weise das folgende XML-Dokument auf (das gleich auch einen Namensraum enthält):

```
<products xmlns="http://www.addison-wesley.de/codebook/csharp">
  <product id="1001">
    <name>Basilikum-Tofu</name>
    <price>2.50</price>
  </product>
  <product id="1002">
    <name>Haselnuss-Tofu</name>
    <price>2.45</price>
  </product>
</products>
```


Sehr hilfreich ist, dass prinzipiell alle konkreten Klassen des X-DOM einen Konstruktor besitzen, über den Sie den Namen und optional auch den Wert des Objekts definieren können:

```
// Namensraum definieren
XNamespace ns = "http://www.addison-wesley.de/codeproduct/csharp";

// Das Wurzelelement erzeugen
XElement rootElement = new XElement(ns + "products");

// Den Namensraum als Attribut angeben
rootElement.Add(new XAttribute("xmlns", ns));

// Das erste Produkt
XElement productElement = new XElement(ns + "product");
productElement.Add(new XAttribute("id", "1001"));
productElement.Add(new XElement(ns + "name", "Basilikum-Tofu"));
productElement.Add(new XElement(ns + "price", 2.5M));
rootElement.Add(productElement);

// Das zweite Produkt
productElement = new XElement(ns + "product");
productElement.Add(new XAttribute("id", "1002"));
productElement.Add(new XElement(ns + "name", "Haselnuss-Tofu"));
productElement.Add(new XElement(ns + "price", 2.45M));
rootElement.Add(productElement);
```

Listing 307: Dynamisches Erzeugen eines XML-Dokuments mit Namensraum auf die klassische Art

Hinweis

Dass Sie am Konstruktor den genauen Typ übergeben, der in dem XML-Dokument verwaltet werden soll, ist sehr wichtig. Die hier übergebenen Daten werden gemäß der XML Schema Definition (XSD) in ihre korrekte String-Repräsentation umgewandelt. Konvertieren Sie diese Werte *nicht* selbst in einen String. Dies würde bei Zahl- und Datumswerten zu ungültigen XML-Daten führen.

Funktionale Konstruktion

Die von Microsoft etwas hochtrabend benannte »Funktionale Konstruktion«[7] ermöglicht die Erstellung eines XML-Elements in einer einzigen Anweisung:

```
XNamespace ns = "http://www.addison-wesley.de/C#-Kompendium/samples";
XElement rootElement =
   new XElement(ns + "products",
      new XElement(ns + "product",
         new XAttribute("id", "1001"),
         new XElement(ns + "name", "Basilikum-Tofu"),
         new XElement(ns + "price", 2.50M)),
      new XElement(ns + "product",
         new XAttribute("id", "1002"),
         new XElement(ns + "name", "Haselnuss-Tofu"),
         new XElement(ns + "price", 2.45M)));
```

Listing 308: Erzeugen eines XML-Dokuments mit der so genannten »Funktionalen Konstruktion«

Hinter dieser Technik steckt allerdings nichts Magisches: Die konkreten X-DOM-Klassen besitzen Konstruktoren, denen am zweiten Argument ein Objekt, oder, bei von `XContainer` abgeleiteten Klassen, auch beliebig viele Objekte als Inhalt übergeben werden können. Damit können Sie die Erzeugung eines ganzen XML-Dokuments in einer Anweisung programmieren.

Sehr einfach zu verstehen ist eine Anweisung mit dieser Variante allerdings nicht. Achten Sie darauf, dass Sie die einzelnen Ebenen des XML-Dokuments wie im Beispiel einrücken, damit das Ganze einigermaßen verständlich bleibt.

Die Funktionale Konstruktion ist sehr hilfreich bei der Abfrage von Daten mit LINQ, für den Fall, dass Sie diese dynamisch in ein XML-Dokument transformieren müssen. Dazu setzen Sie die funktionale Konstruktion im Selektor der Abfrage ein. Diese Technik setze ich in 185 ein.

Ändern

Das Ändern von XML-Dokumenten über das X-DOM ist sehr einfach: Suchen Sie über LINQ die zu ändernden Elemente und ändern Sie deren Daten. Dabei sollten Sie aber nur bei reinen String-Feldern die `Value`-Eigenschaft verwenden. Bei allen anderen Typen rufen Sie zum Schreiben von Daten die `SetValue`-Methode auf. `SetValue` konvertiert übergebene Werte und Objekte automatisch in das Format, das die XML Schema Definition (XSD) vorschreibt.

Beachten Sie, dass ich im Beispiel für das *price*-Element den Namensraum angebe, da ansonsten kein Zugriff möglich wäre.

7. Der Begriff »Funktionale Konstruktion« soll an die funktionale Programmierung anknüpfen. In meinen Augen hat die klassische Anwendung von Konstruktoren und `params`-Argumenten mit der funktionalen Programmierung aber rein gar nichts zu tun.

```
// Artikel suchen
var product1 = rootElement.Elements().Single(
   p => p.Attribute("id").Value == "1001");
if (product1 != null)
{
   // Artikel ändern
   product1.Element(ns + "price").SetValue(2.55M);
}
```

Listing 309: Ändern eines XML-Dokuments mit LINQ to SQL und dem X-DOM

Löschen

Das Löschen ist genauso einfach wie das Ändern: Sie suchen das zu löschende Element und rufen dessen `Remove`-Methode auf:

```
// Artikel suchen
var product2 = rootElement.Elements().Single(
   p => p.Attribute("id").Value == "1002");
if (product2 != null)
{
   // Artikel löschen
   product2.Remove();
}
```

Listing 310: Löschen eines Elements

Remove löscht das komplette Element, inklusive aller Kind-Knoten.

180 XML-Dokumente über LINQ to XML erzeugen

Das Erzeugen eines XML-Dokuments mit LINQ ist sehr einfach, weil die speziellen Konstruktoren der Klasse `XElement` das Erzeugen in einer Anweisung erlauben. So können Sie beliebige Daten abfragen (alle Auflistungen, Datenbanken und sonstige Daten, für die eine LINQ-Erweiterung zu Verfügung steht) und aus diesen ein XML-Dokument erzeugen. Das Prinzip ist einfach und basiert auf den LINQ-Grundlagen und der funktionalen Konstruktion (Rezept 179).

Die Besonderheit ist allerdings, dass Sie die LINQ-Abfrage als Inhalt eines `XElement`-Objekts angeben können. Damit können Sie z. B. ein Wurzelelement mit Kind-Elementen füllen, die Sie aus einer beliebigen Datenquelle auslesen.

Das folgende Beispiel fragt auf diese Weise Artikeldaten aus einer Auflistung in ein XML-Dokument ab und speichert dieses. Ein wesentlicher Trick ist, dass das Beispiel ermittelt, ob das Feld `CategoryID` der aufgelisteten `Product`-Objekte `null` ist und in diesem Fall im erzeugten XML-Dokument ein korrekt mit `xsi:nil="true"` definiertes XML-Element anlegt.

```
/* Demo-Klasse */
public class Product
{
   public string Name;
   public Decimal Price;
   public int? CategoryID;
}

...

// Daten "einlesen"
Product[] products = {
   new Product { Name = "Neill Pryde Excess 5.4", Price = 500,
      CategoryID = 1},
   new Product { Name = "Neill Pryde Excess 6.4", Price = 510,
      CategoryID = 1},
   new Product { Name = "Chinook Carbon Mast Extension", Price = 125}};

// Die Daten in ein XML-Dokument projizieren
XNamespace xsi = XmlSchema.InstanceNamespace;
var productsElement = new XElement("products",
   new XAttribute(XNamespace.Xmlns + "xsi", xsi),
   from product in products
   select new XElement("product",
      new XElement("name", product.Name),
      new XElement("category", product.CategoryID,
         product.CategoryID == null ?
         new XAttribute(xsi + "nil", true) : null)));
```

Listing 311: Projektion in ein XML-Dokument mit einem bedingungsabhängigen Hinzufügen des xsi:nil-Attributs zu einem Element

Das Ergebnis dieses Beispiels ist:

```
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <product>
    <name>Neill Pryde Excess 5.4</name>
    <category>1</category>
  </product>
  <product>
    <name>Neill Pryde Excess 6.4</name>
    <category>1</category>
  </product>
  <product>
    <name>Chinook Carbon Mast Extension</name>
    <category xsi:nil="true" />
  </product>
</products>
```

Dieses Feature ist deswegen möglich, weil die Konstruktoren von `XElement` an `Object`-Referenzen auch `IEnumerable<T>`-Auflistungen verarbeiten und die einzelnen Objekte in dieser Auflistung als Kind-Elemente anlegen. Dies ist in der Praxis sehr hilfreich. Es bedeutet auch, dass Sie LINQ-Abfragen bei der funktionalen Konstruktion von XML-Dokumenten ebenfalls in tieferen Ebenen des Dokuments einsetzen können. Das macht das Erzeugen eines XML-Dokuments nicht gerade übersichtlicher, aber in der Praxis wesentlich einfacher als mit anderen Mitteln.

Der Trick in diesem Beispiel ist der letzte Teil:

```
new XElement("category", product.CategoryID,
    product.CategoryID == null ?
    new XAttribute(xsi + "nil", true) : null)
```

Hier wird ein XML-Element mit dem Namen *category* angelegt, das den Wert erhält, der in dem Feld `CategoryID` des jeweiligen `Product`-Objekts gespeichert ist. Da das `CategoryID`-Feld aber auch `null` sein kann, würde LINQ to XML ein leeres Feld erzeugen. Um dieses korrekt mit dem XSD-Attribut `nil` zu kennzeichnen, wird am dritten Argument des `XElement`-Konstruktors (das weitere Inhalte des Elements aufnimmt) über den Bedingungsoperator geprüft, ob `CategoryID` `null` ist, und in diesem Fall ein `XAttribute` für das XSD-Attribut `nil` angefügt.

181 XML-Dokumente über ein DataTable- oder DataSet-Objekt erzeugen

`DataTable`- und `DataSet`-Objekte (aus dem Namensraum `System.Data`) erlauben das sehr einfache Erzeugen und Verändern von XML-Dokumenten, die eine tabellenähnliche (auf zwei Ebenen beschränkte) Struktur aufweisen. Zum Erzeugen müssen Sie lediglich das `DataTable`- bzw. das `DataSet`-Objekt dynamisch im Speicher erzeugen, mit Daten füllen und schließlich die `WriteXml`-Methode aufrufen um die XML-Datei zu schreiben.

Der Unterschied zwischen der Verwendung eines `DataTable`- und eines `DataSet`-Objekts zum Lesen ist, dass Sie in einem `DataTable`-Objekt nur XML-Dokumente verwalten können, die aus einer »Tabelle« bestehen. In einem `DataSet`-Objekt können Sie auch solche Dokumente verwalten, die mehrere »Tabellen« beinhalten.

Um den Namen des Wurzel-Elements festzulegen, können Sie nach der Erzeugung eines `DataSet` diesen der Eigenschaft `DataSetName` zuweisen. Einem `DataTable`-Objekt können Sie scheinbar leider nicht den Namen des Wurzel-Elements mitteilen. Den zu verwendenden Namensraum und den Namensraum-Präfix können Sie über die Eigenschaften `Namespace` und `Prefix` festlegen.

Der Programmcode in Listing 312 erzeugt eine der im Rezept 171 eingesetzten XML-Datei ähnliche Datei zur Speicherung von Personendaten. Nach der Erzeugung des `DataSet`-Objekts fügt das Programm diesem ein `DataTable`-Objekt an und definiert die einzelnen Spalten der Tabelle. Über die `Rows`-Auflistung werden dann einzelne Personen-Datensätze hinzugefügt. Beim Schreiben der XML-Datei übergibt das Programm am zweiten Argument der `WriteXml`-Methode den Wert `XmlWriteMode.WriteSchema`, um dem Dokument ein eingebettetes Schema hinzuzufügen. Dabei entstehende Ausnahmen werden abgefangen.

Um zu verhindern, dass eine bereits existierende Datei ohne Warnung überschrieben wird, fragt das Programm zu Beginn ab, ob eine solche existiert, und lässt den Anwender entscheiden, ob die Datei überschrieben werden soll.

Das Beispiel erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Xml` und `System.Windows.Forms`.


```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// Überprüfen, ob die Datei existiert
if (File.Exists(xmlFileName))
{
   if (MessageBox.Show("Die Datei '" + xmlFileName +
      "' existiert bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
      Application.ProductName, MessageBoxButtons.YesNo,
      MessageBoxIcon.Question) == DialogResult.No)
   {
      return;
   }
}

// DataSet erzeugen und initialisieren
DataSet dataSet = new DataSet();
dataSet.DataSetName = "persons";
dataSet.Namespace = "http://www.addison-wesley.de/codebook/persons";
dataSet.Prefix = "per";

// Tabelle erzeugen
DataTable dataTable = dataSet.Tables.Add("person");

// Spalten definieren
dataTable.Columns.Add("id", typeof(int));
dataTable.Columns.Add("firstname", typeof(string));
dataTable.Columns.Add("lastname", typeof(string));
dataTable.Columns.Add("type", typeof(string));

// Zeilen anfügen
dataTable.Rows.Add(new object[] {1000, "Zaphod", "Beeblebrox", "Alien"});
dataTable.Rows.Add(new object[] {1001, "Ford", "Prefect", "Alien"});
dataTable.Rows.Add(new object[] {1002, "Tricia", "McMillan", "Earthling"});
dataTable.Rows.Add(new object[] {1003, "Arthur", "Dent", "Earthling"});

// XML-Datei mit Schema schreiben
try
```

Listing 312: Erzeugen eines XML-Dokuments über ein DataSet-Objekt

```
{
   dataSet.WriteXml(xmlFileName, XmlWriteMode.WriteSchema);
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Schreiben der XML-Datei: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   return;
}
```

Listing 312: Erzeugen eines XML-Dokuments über ein DataSet-Objekt (Forts.)

182 XML-Dokumente über ein DataTable- oder ein DataSet-Objekt ändern

Das Verändern eines XML-Dokuments, das eine tabellenähnliche Struktur (mit Elementen in zwei Ebenen) aufweist, ist über ein `DataTable`- oder ein `DataSet`-Objekt sehr einfach, da Sie dazu die Methoden und Eigenschaften der `DataSet`- und der `DataTable`-Klasse verwenden können. Das einzige Problem ist, dass die XML-Datei eine Struktur aufweisen muss, die vom `DataTable`- bzw. vom `DataSet`-Objekt gelesen werden kann. Idealerweise verwenden Sie XML-Dokumente, die über ein solches Objekt erzeugt wurden.

Das folgende Listing zeigt dies am Beispiel der Personen-XML-Datei, die im Rezept 181 erstellt wird. In diesem Beispiel werden eine Person hinzugefügt, die Person mit der ID 1003 gelöscht und die Daten der Person mit der ID 1001 geändert.

Zum Kompilieren des Beispiels müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Data`, `System.IO` und `System.Windows.Forms` importieren.

```
// Der Dateiname der XML-Datei
string xmlFileName = Path.Combine(Application.StartupPath, "Persons.xml");

// DataSet erzeugen und XML-Datei mit Schema einlesen
DataSet dataSet = new DataSet();
try
{
   dataSet.ReadXml(xmlFileName, XmlReadMode.ReadSchema);
}
catch (Exception ex)
{
   MessageBox.Show("Fehler beim Lesen der XML-Datei: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   return;
}
```

Listing 313: Verändern einer XML-Datei über ein DataSet-Objekt

```
// DataTable referenzieren
DataTable personTable = dataSet.Tables["person"];

if (personTable == null)
{
   MessageBox.Show("Das XML-Dokument enthält keine person-Elemente",
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   return;
}

// Zeile anfügen
personTable.Rows.Add(new Object[] {1004, "Marvin", "", "Robot"});

// Zeile suchen und löschen
DataRow[] rows = personTable.Select("id = 1003");
for (int i = 0; i < rows.Length; i++)
{
   rows[i].Delete();
}

// Zeile suchen und aktualisieren
rows = personTable.Select("id = 1001");
if (rows.Length == 1)
{
   rows[0]["type"] = "Traveller";
}
else
{
   if (rows.Length == 0)
   {
      Console.WriteLine("Person mit der Id 1001 nicht gefunden");
   }
   else
   {
      Console.WriteLine("Mehrere Personen mit der Id 1001 gefunden");
   }
}

// XML-Datei mit Schema (unter einem anderen Namen) schreiben
try
{
   xmlFileName = xmlFileName.Replace("Persons", "Persons_New");
   dataSet.WriteXml(xmlFileName, XmlWriteMode.WriteSchema);
}
catch (Exception ex)
```

Listing 313: Verändern einer XML-Datei über ein DataSet-Objekt (Forts.)

```
{
   MessageBox.Show("Fehler beim Schreiben der XML-Datei: " + ex.Message,
      Application.ProductName, MessageBoxButtons.OK, MessageBoxIcon.Error);
   return;
}
```

Listing 313: Verändern einer XML-Datei über ein DataSet-Objekt (Forts.)

183 Binäre Daten in einem XML-Dokument speichern

In XML-Dokumenten können Sie nicht nur Textdaten speichern, sondern auch binäre. Das Prinzip dazu ist ganz einfach: Sie müssen die binären Daten lediglich über das Base64-Verfahren in einen Text umwandeln, der im 7-Bit-ASCII-Zeichensatz codiert ist. Dazu können Sie beim Schreiben über ein `XmlWriter`-Objekt die Methode `WriteBase64` verwenden. Verwenden Sie ein `XElement`- oder `XDocument`-Objekt, müssen Sie die Daten über `Convert.ToBase64String` selbst umwandeln.

Das Beispiel in Listing 314 erzeugt ein neues XML-Dokument mit einem `person`-Element, in dem neben dem Vor- und dem Nachnamen auch ein Bild der Person gespeichert wird.

Das Programm erfordert den Import der Namensräume `System`, `System.Diagnostics`, `System.IO`, `System.Reflection` und `System.Xml.Linq`.

```
// Der Dateiname
string xmlFileName = "Persons.xml";

// Neues XML-Dokument erzeugen
XElement rootElement = XElement.Parse(
   @"<persons>
      <person>
         <firstName>Zaphod</firstName>
         <lastName>Beeblebrox</lastName>
         <picture/>
      </person>
   </persons>");

// Das Bild einlesen
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string imageFileName = Path.Combine(appPath, "Zaphod.jpg");
FileInfo fi = new FileInfo(imageFileName);
byte[] buffer = new byte[fi.Length];
FileStream fs = new FileStream(imageFileName, FileMode.Open, FileAccess.Read);
fs.Read(buffer, 0, buffer.Length);
```

Listing 314: Speichern von binären Daten in einer XML-Datei

```
// Die Person suchen
XElement personElement = rootElement.Element("person");

// Bild Base64-codieren ...
string codedFile = Convert.ToBase64String(buffer, 0, buffer.Length);

// Das Bild schreiben
personElement.Element("picture").Value = codedFile;

// XML-Datei speichern
rootElement.Save(xmlFileName);
```

Listing 314: Speichern von binären Daten in einer XML-Datei (Forts.)

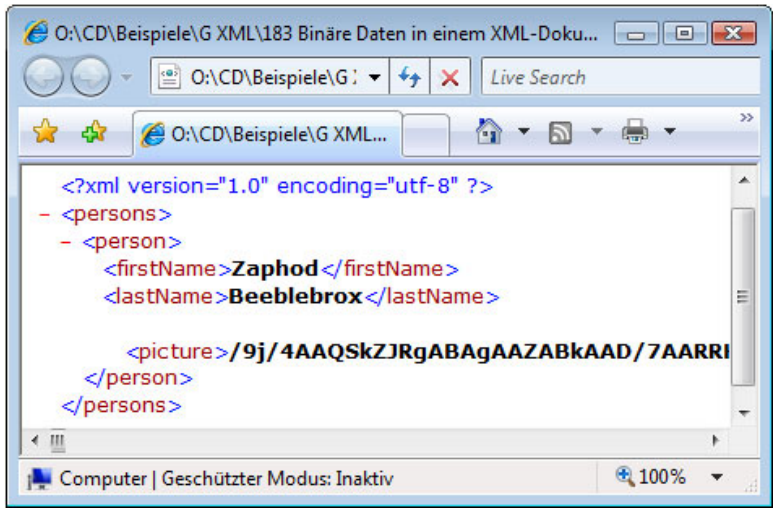


Abbildung 100: Die erzeugte XML-Datei im Internet Explorer

184 Base64-codierte Bilder (binäre Daten) aus XML-Dokumenten auslesen

Binäre Daten, die im Base64-Format in einem XML-Dokument gespeichert sind, können Sie über die Methode `FromBase64String` der `Convert`-Klasse in ein `byte`-Array konvertieren. Dieses Array können Sie dann über ein `Stream`-Objekt weiterverarbeiten (z. B. über einen `FileStream` in eine Datei schreiben).

Das (Windows.Forms-)Beispiel in Listing 315 liest das Bild aus der im vorhergehenden Rezept erzeugten XML-Datei ein und erzeugt damit ein `MemoryStream`-Objekt. Dieses Objekt wird dem Konstruktor der `Bitmap`-Klasse übergeben um ein `Bitmap`-Objekt zu erzeugen. Die Referenz auf das `Bitmap`-Objekt wird dann in die `Image`-Eigenschaft eines `PictureBox`-Steuerelements geschrieben.

Zum Kompilieren des Beispiels müssen Sie die Namensräume `System`, `System.Drawing`, `System.IO`, `System.Linq` und `System.Xml.Linq` importieren.

```
// XML-Datei einlesen
string xmlFileName = "Persons.xml";
XElement rootElement = XElement.Load(xmlFileName);
XElement personElement = rootElement.Element("person");
XElement pictureElement = personElement.Element("picture");

// Base64-codierten String in Bitmap umwandeln
byte[] buffer = Convert.FromBase64String(pictureElement.Value);
Bitmap bitmap = new Bitmap(new MemoryStream(buffer,
   0, buffer.Length));
```

Listing 315: Auslesen eines in einer XML-Datei gespeicherten Bilds

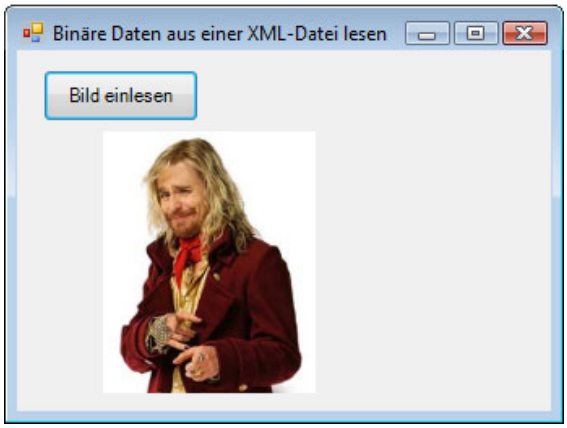


Abbildung 101: Das (Windows.Forms-)Beispiel zu diesem Rezept hat das Bild aus der XML-Datei eingelesen und einer PictureBox zugewiesen

185 XML-Dokumente über LINQ to XML transformieren

Über die in Rezept 180 besprochene Technik können Sie XML-Dokumente auch sehr gut in ein anderes XML-Format transformieren (z. B. in XHTML). Das Vorgehen entspricht dem Transformieren anderer Daten in ein XML-Dokument, ist aber für die Praxis sehr interessant. In dieser kommt es nämlich immer wieder vor, dass die Struktur eines XML-Dokuments, das in einer Anwendung verwendet wird, geändert wird, wenn die Anwendung weiterentwickelt wird. Wenn eine neue Version der Anwendung dann beim Kunden ggf. XML-Dokumente einlesen muss, die in der alten Struktur vorliegen, müssen diese transformiert werden. Das gilt z. B. auch dann, wenn Sie mit einer Serialisierung arbeiten und die serialisierten Typen in neuen Versionen der Anwendung ändern.

Zur Transformation können Sie das sehr komplexe XSLT verwenden. Sie können die Transformation aber auch einfach über LINQ to XML vornehmen. Der Programmcode, der dabei herauskommt, sieht für externe Programmierer zwar auch nicht besonders übersichtlich aus. Für den Programmierer, der den Programmcode entwickelt hat, ist der Aufbau aber relativ logisch.

Das folgende Beispiel soll das Prinzip verdeutlichen. Angenommen, eine erste Version der Anwendung speichert Artikeldaten in einem einfachen XML-Dokument:


```
<?xml version="1.0" encoding="utf-8" ?>
<products>
   <product>
      <id>1001</id>
      <name>Neill Pryde Excess 5.4</name>
   </product>
   <product>
      <id>1002</id>
      <name>Neill Pryde Excess 6.4</name>
   </product>
</products>
```

Die neue Version der Anwendung erwartet die Artikel-ID nicht in einem Element, sondern einem Attribut. Außerdem erwartet diese das Element `price`, das allerdings Nullwerte erlaubt:

```
<?xml version="1.0" encoding="utf-8" ?>
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <product id="1001">
      <name>Neill Pryde Excess 5.4</name>
      <price xsi:nil="true"/>
   </product>
   <product id="1002">
      <name>Neill Pryde Excess 6.4</name>
      <price xsi:nil="true"/>
   </product>
</products>
```

Die Transformation über LINQ to XML ist sehr gradlinig:

```
// Einlesen der alten Version
XElement productsElement = XElement.Load("Products.xml");

// Konvertieren dieser Version in eine neue
XNamespace xsi = XmlSchema.InstanceNamespace;
XElement newProductsElement =
   new XElement("products",
   new XAttribute(XNamespace.Xmlns + "xsi", xsi),
   from productElement in productsElement.Elements("product")
   select new XElement("product",
      new XAttribute("id", productElement.Element("id").Value),
```

Listing 316: Transformieren eines XML-Dokuments in ein anderes Format über LINQ to XML

```
      new XElement("name", productElement.Element("name").Value),
      new XElement("price",
         new XAttribute(xsi + "nil", "true"),
         null)));
```

Listing 316: Transformieren eines XML-Dokuments in ein anderes Format über LINQ to XML (Forts.)

186 XML-Dokumente über XSL-Dokumente transformieren

XML-Dokumente müssen recht häufig in eine andere Form transformiert werden (z. B. in XHTML). Üblicherweise wird dazu XSL (Extensible Stylesheet Language) verwendet.

Hinweis

Mit LINQ to XML haben Sie auch noch die in Rezept 185 besprochene Möglichkeit, XML-Dokumente über LINQ zu transformieren, die in der Praxis häufig einfacher ist.

XSL ist eine Sprache mit einer Vielzahl an speziellen Instruktionen, über die Sie die Daten eines XML-Dokuments sehr flexibel auslesen, transformieren und in ein Ergebnis-Dokument speichern können. Ein einfaches XSL-Dokument, das die Personen-XML-Datei aus dem Rezept 171 in eine XHTML-Datei transformiert, sieht z. B. so aus:

```
<?xml version="1.0"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">

<!-- Transformation nach HTML -->

<body>
<h1>Personen</h1>
<p>

<xsl:for-each select = 'persons/person'>
  <xsl:sort select='lastname' order='ascending'/>
  <xsl:value-of select='lastname'/>,
  <xsl:value-of select='firstname'/>:
  <xsl:value-of select='type'/>
<br/>

</xsl:for-each>
</p>

</body>

</html>
```

Listing 317: Ein XSL-Dokument zur Transformation der Personen-XML-Datei nach XHTML

Das (etwas unschöne) aus den transformierten XML-Daten erzeugte HTML-Dokument sieht dann folgendermaßen aus:

```
<html>
  <body>
    <h1>Personen</h1>
    <p>Beeblebrox,
  Zaphod:
  Alien<br>Dent,
  Arthur:
  Earthling<br>McMillan,
  Tricia:
  Earthling<br>Prefect,
  Ford:
  Alien<br></p>
  </body>
</html>
```

Listing 318: Die transformierte Personen-XML-Datei

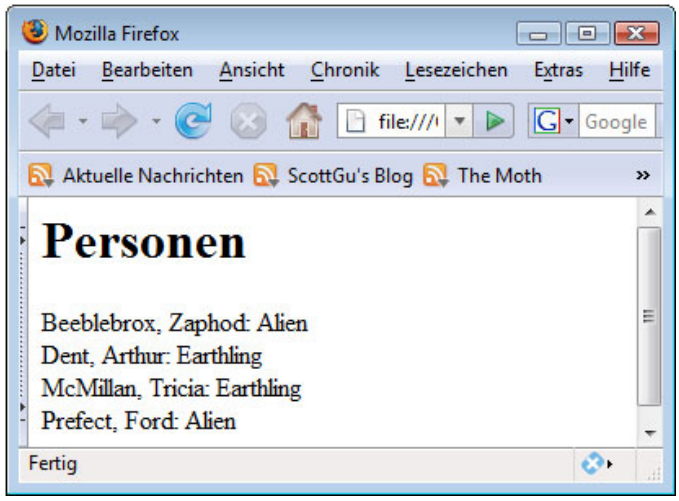


Abbildung 102: Die erzeugte HTML-Datei in einem Webbrowser

Eine solche Transformation können Sie über eine Instanz der Klasse `XslCompiledTransform` aus dem Namensraum `System.Xml.Xsl` vornehmen.

Die Methode `TransformXmlFile` in Listing 319 erzeugt aus einem XML-Dokument über ein XSL-Dokument eine Ergebnisdatei. Die entsprechenden Dateinamen werden dieser Methode übergeben. `TransformXmlFile` erzeugt zunächst eine Instanz der `XslTransform`-Klasse und lädt die XSL-Datei über deren `Load`-Methode. Die Transformation wird dann über die Methode `Transform` ausgeführt.

`TransformXmlFile` erfordert den Import der Namensräume `System`, `System.Xml` und `System.Xml.Xsl`.

```
public static void TransformXmlFile(string xmlSourceFileName,
   string xslFileName, string xmlDestFileName)
{
   // Instanz der Klasse XslTransform erzeugen
   XslCompiledTransform xslTransform = new XslCompiledTransform();

   // XSL-Datei laden
   xslTransform.Load(xslFileName);

   // XML-Datei transformieren
   xslTransform.Transform(xmlSourceFileName, xmlDestFileName);
}
```

Listing 319: Methode zur Transformation einer XML-Datei

System

187 Den Namen des Computers ermitteln

Den Namen des Computers, auf dem ein .NET-Programm ausgeführt wird, können Sie über die Eigenschaft `MachineName` der `Environment`-Klasse aus dem Namensraum `System` auslesen:

```
string machineName = System.Environment.MachineName;
```

188 Systempfade herausfinden

Der Pfad zum Windows-System-Ordner steht Ihnen über die Eigenschaft `SystemDirectory` der Klasse `System.Environment` zur Verfügung. Eine Möglichkeit, das Windows-Verzeichnis selbst auszulesen, fehlt, wahrscheinlich deshalb, weil die `Environment`-Klasse betriebssystemübergreifend funktionieren soll. Den Pfad zum Windows-Verzeichnis können Sie aber recht einfach über die API-Funktion `GetWindowsDirectory` auslesen. Diese Funktion erwartet am ersten Argument normalerweise einen Zeiger auf einen ausreichend groß dimensionierten (C-)String. In C# können Sie an diesem Argument ein `StringBuilder`-Objekt übergeben, das Sie zuvor mit der maximal möglichen Anzahl Zeichen (bei Pfaden zurzeit 260) initialisieren müssen. Das Beispielprogramm initialisiert die Größe auf ein Zeichen mehr, da die API-Funktion den String mit einem 0-Zeichen abschließt. Die Größe dieses Puffers übergeben Sie `GetWindowsDirectory` am zweiten Argument.

Über die Methode `GetFolderPath` der `Environment`-Klasse können Sie den Pfad zu weiteren Systemordnern ermitteln. Diese Methode erwartet als Argument einen Wert der Aufzählung `Environment.SpecialFolder` (Tabelle 20).

Konstante	Bedeutung
`ApplicationData`	Pfad zum Verzeichnis für programmspezifische Daten für Benutzer, deren Profil in einem Netzwerk-Ordner gespeichert und beim Einloggen auf den lokalen Computer kopiert wird (»wandernde Benutzer«)
`CommonApplicationData`	Pfad zum Verzeichnis für programmspezifische Daten für alle Benutzer
`CommonProgramFiles`	Pfad zum Verzeichnis für Komponenten, die von mehreren Anwendungen verwendet werden
`Cookies`	Pfad zum Verzeichnis, in dem Cookies gespeichert werden
`DesktopDirectory`	Pfad zum Desktop-Verzeichnis des aktuellen Benutzers
`Favorites`	Pfad zum Favoriten-Verzeichnis des aktuellen Benutzers
`History`	Pfad zum Internet-Verlaufs-Verzeichnis des aktuellen Benutzers
`InternetCache`	Pfad zum Verzeichnis für temporäre Internetdateien
`LocalApplicationData`	Pfad zum Verzeichnis für programmspezifische Daten für Benutzer, deren Profil lokal verwaltet wird (»nicht wandernde Benutzer«)
`Personal`	Pfad zum Verzeichnis für die eigenen Dateien des aktuellen Benutzers

Tabelle 20: Die Werte der Aufzählung SpecialFolder

Konstante	Bedeutung
ProgramFiles	Pfad zum Verzeichnis, in dem per Voreinstellung alle Programme installiert werden
Programs	Pfad zum Verzeichnis, in dem die Programmgruppen des aktuellen Benutzers verwaltet werden
Recent	Pfad zum Verzeichnis, in dem Verweise auf die zuletzt geöffneten Dokumente für den aktuellen Benutzer verwaltet werden
SendTo	Pfad zum Verzeichnis, in dem die Verweise verwaltet werden, die im Senden-An-Kontextmenüeintrag erscheinen
StartMenu	Pfad zum Verzeichnis, in dem die Startmenüeinträge des aktuellen Benutzers verwaltet werden
Startup	Pfad zum Autostart-Verzeichnis
System	Pfad zum Windows-System32-Verzeichnis
Templates	Pfad zum Verzeichnis, in dem Dokumentenvorlagen verwaltet werden

Tabelle 20: Die Werte der Aufzählung SpecialFolder (Forts.)

Listing 320 liest einige Systempfade und gibt diese an der Konsole aus. Das Programm benötigt den Import der Namensräume `System`, `System.Text` und `System.Runtime.InteropServices`.

```
/* Deklaration der API-Funktion GetWindowsDirectory */
[DllImport("kernel32.dll")]
private static extern uint GetWindowsDirectory(StringBuilder lpBuffer,
   uint uSize);

...

// Windows-Systempfad auslesen
string windowsSystemPath = System.Environment.SystemDirectory;

// Windows-Pfad auslesen
const int MAX_PATH = 260;
StringBuilder buffer = new StringBuilder(MAX_PATH + 1);
string windowsPath = null;
if (GetWindowsDirectory(buffer, MAX_PATH + 1) > 0)
   windowsPath = buffer.ToString();

// Spezielle Systemordner ermitteln
string programPath = Environment.GetFolderPath(
   Environment.SpecialFolder.ProgramFiles);
string desktopPath = Environment.GetFolderPath(
   Environment.SpecialFolder.DesktopDirectory);
string nonRoamingUserApplicationDataPath = Environment.GetFolderPath(
   Environment.SpecialFolder.LocalApplicationData);
```

Listing 320: Auslesen einiger Windows-Systempfade

```
string roamingUserApplicationDataPath = Environment.GetFolderPath(
   Environment.SpecialFolder.ApplicationData);
string allUsersApplicationDataPath = Environment.GetFolderPath(
   Environment.SpecialFolder.CommonApplicationData);
string recentPath = Environment.GetFolderPath(
   Environment.SpecialFolder.Recent);

Console.WriteLine("Windows-Pfad:\r\n{0}", windowsPath);
Console.WriteLine("\r\nWindows-System-Pfad:\r\n{0}", windowsSystemPath);
Console.WriteLine("\r\nProgrammordner:\r\n{0}", programPath);
Console.WriteLine("\r\nDaten-Ordner für nicht wandernde Benutzer:\r\n{0}",
   nonRoamingUserApplicationDataPath);
Console.WriteLine("\r\nDaten-Ordner für wandernde Benutzer:\r\n{0}",
   roamingUserApplicationDataPath);
Console.WriteLine("\r\nDaten-Ordner für alle Benutzer:\r\n{0}",
   allUsersApplicationDataPath);
Console.WriteLine("\r\nDesktop:\r\n{0}", desktopPath);
Console.WriteLine("\r\nZuletzt geöffnete Dokumente:\r\n{0}", recentPath);
```

Listing 320: Auslesen einiger Windows-Systempfade (Forts.)

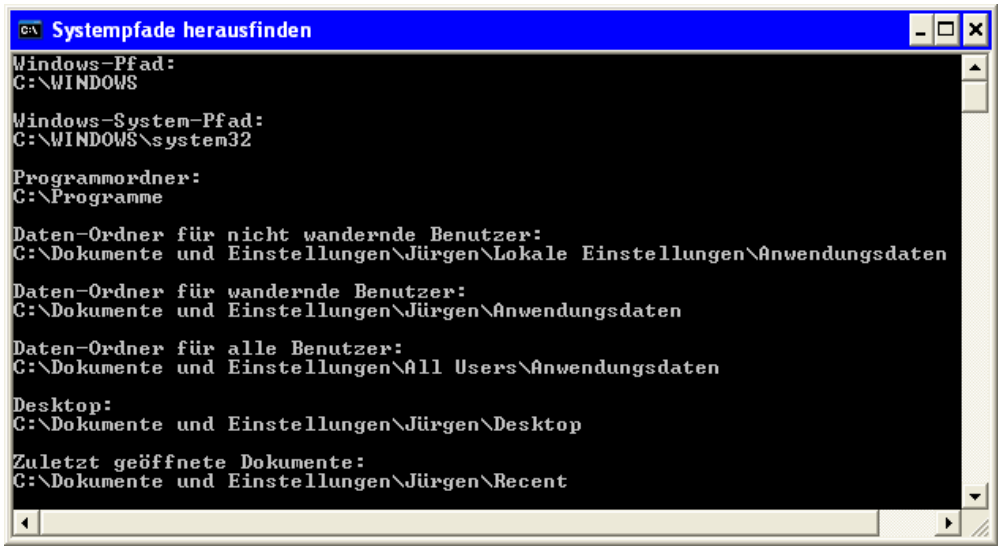


Abbildung 103: Das Programm zum Auslesen einiger Systempfade

Hinweis

In Rezept 112, das dieses Thema ebenfalls behandelt, habe ich Methoden entwickelt, die die in meinen Augen wichtigsten Systempfade zurückliefern.

189 Windows-Version auslesen

Die Version des aktuellen Betriebssystems können Sie über das `Version`-Objekt auslesen, das `System.Environment.OSVersion.Version` referenziert. Die einzelnen Teile der Version stehen über die Eigenschaften `Major`, `Minor`, `Revision` und `Build` zur Verfügung. Die Eigenschaft `Platform` liefert einen String mit der Kurzbezeichnung des Betriebssystems. Für die NT-Linie von Microsoft gibt diese Eigenschaft den String "Win32NT" zurück.

```
using System;

...

Version version = Environment.OSVersion.Version;
PlatformID platformId = Environment.OSVersion.Platform;

Console.WriteLine("Version: Major: {0}", version.Major);
Console.WriteLine("Version: Minor: {0}", version.Minor);
Console.WriteLine("Version: Revision: {0}", version.Revision);
Console.WriteLine("Version: Build: {0}", version.Build);
Console.WriteLine("Plattform: {0}", platformId);
```

Listing 321: Ermitteln der Windows-Version und der Plattform

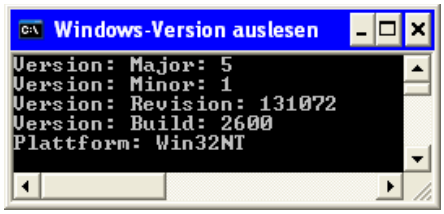


Abbildung 104: Ermitteln der Windows-Version auf einem Windows-XP-Rechner

Dummerweise ist es über diese Klasse nicht allzu einfach, die genaue Version des Betriebssystems herauszufinden. Über die Versionsnummer können Sie zwar herausfinden, auf welcher Windows-Version das Programm ausgeführt wird. Windows 2000 besitzt z. B. die Versionsnummer 5.0.x.x, Windows XP die Versionsnummer 5.1.x.x. Im nächsten Rezept finden Sie eine Lösung zur Ermittlung der Windows-Hauptversion.

190 Windows-Hauptversion ermitteln

Die Methode `GetWindowsMainVersion` in Listing 322 ermittelt die Windows-Hauptversion basierend auf den Angaben, die Microsoft an der Adresse *msdn.microsoft.com/en-us/library/bb968799(VS.85).aspx* veröffentlicht. Um die Version in einer flexibel verarbeitbaren Form zurückzugeben, habe ich zunächst eine Aufzählung `WindowsMainVersion` implementiert, die die möglichen Windows-Versionen als Konstanten beinhaltet. Die Konstante `Unknown` steht dafür, dass die Version nicht ermittelt werden konnte (weil es sich z. B. um eine neuere Version handelt).

Zum Kompilieren des Quellcodes müssen Sie den Namensraum `System` importieren.

```
/* Aufzählung für die Windows-Hauptversion */
public enum WindowsMainVersion
{
   Windows95,
   Windows98,
   WindowsMe,
   WindowsNT4,
   Windows2000,
   WindowsXP,
   WindowsServer2003OrXP64,
   Vista,
   Unknown
}

/* Ermittelt die Windows-Hauptversion */
public static WindowsMainVersion GetWindowsMainVersion()
{
   switch (Environment.OSVersion.Version.Major)
   {
      case 4:
         if (Environment.OSVersion.Version.Minor == 0)
         {
            if (Environment.OSVersion.Platform ==
               PlatformID.Win32NT)
            {
               return WindowsMainVersion.WindowsNT4;
            }
            else
            {
               return WindowsMainVersion.Windows95;
            }
         }
         else if (Environment.OSVersion.Version.Minor == 10)
         {
            return WindowsMainVersion.Windows98;
         }
         else if (Environment.OSVersion.Version.Minor == 90)
         {
            return WindowsMainVersion.WindowsMe;
         }
         else
         {
            return WindowsMainVersion.Unknown;
         }

      case 5:
```

Listing 322: Methode zur Ermittlung der Windows-Hauptversion

```
            if (Environment.OSVersion.Version.Minor == 0)
            {
               return WindowsMainVersion.Windows2000;
            }
            else if (Environment.OSVersion.Version.Minor == 1)
            {
               return WindowsMainVersion.WindowsXP;
            }
            else if (Environment.OSVersion.Version.Minor == 2)
            {
               // Hier kann leider nicht genau entschieden werden,
               // ob es sich um Windows Server 2003 oder XP 64 Bit handelt
               return WindowsMainVersion.WindowsServer2003OrXP64;
            }
            else
            {
               return WindowsMainVersion.Unknown;
            }

         case 6:
            return WindowsMainVersion.Vista;

         default:
            return WindowsMainVersion.Unknown;
      }
   }
```

Listing 322: Methode zur Ermittlung der Windows-Hauptversion (Forts.)

Dummerweise besitzen Windows Server 2003 und Windows XP Professional 64 Bit nach den Informationen von Microsoft dieselbe Haupt- und Nebenversion (5.2) und können deswegen nicht einfach unterschieden werden. Denkbar wäre eine Unterscheidung nach der Bitbreite des Betriebssystems, aber ich habe nicht ermitteln können, wie man diese herausfindet.

191 Die Prozessorgeschwindigkeit ermitteln

Die Geschwindigkeit der Prozessoren eines Computers können Sie über WMI ermitteln. Dazu erzeugen Sie eine Instanz der Klasse `System.Management.ManagementObject`, der Sie im Konstruktor einen String in der Form "Win32_Processor.DeviceID='CPU0'" übergeben. Damit erzeugen Sie eine Instanz der WMI-Klasse `Win32_Processor` für den ersten Prozessor des Systems. Für den zweiten Prozessor müssen Sie als *DeviceID* den Wert *CPU1* angeben. Die Prozessorgeschwindigkeit können Sie dann über den Indexer auslesen, indem Sie diesem als Index den String "CurrentClockSpeed" übergeben. Den zurückgegebenen `object`-Wert sollten Sie anschließend noch in einen `uint`-Wert konvertieren.

Die Methode `GetProcessorSpeed` in Listing 323 macht genau das. Für den Fall, dass die Prozessorgeschwindigkeit nicht verfügbar ist, gibt `GetProcessorSpeed` 0 zurück. Zum Kompilieren

dieser Methode müssen Sie die Assembly *System.Management.dll* referenzieren und die Namensräume `System` und `System.Management` einbinden.

```
public static uint GetProcessorSpeed()
{
   // Win32_Processor-Instanz für den ersten Prozessor erzeugen und die
   // Geschwindigkeit abfragen
   ManagementObject mo = new ManagementObject(
      "Win32_Processor.DeviceID='CPU0'");
   uint currentClockSpeed = 0;
   try
   {
      currentClockSpeed = (uint)(mo["CurrentClockSpeed"]);
   }
   catch {}

   // Speicher des WMI-Objekts freigeben um den Arbeitsspeicher
   // möglichst schnell zu entlasten
   mo.Dispose();

   return currentClockSpeed;
}
```

Listing 323: Methode zur Ermittlung der Prozessorgeschwindigkeit

Das Programm funktioniert nur dann, wenn WMI auf dem Computer, auf dem es ausgeführt wird, installiert ist. Außerdem muss das aktuelle Benutzerkonto über ausreichend WMI- und Systemrechte verfügen. Falls dies nicht der Fall ist, können Sie vor der Ausführung dieser Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 307 beschreibe.

192 Informationen über den Arbeitsspeicher auslesen

Informationen über den Arbeitsspeicher des Systems erhalten Sie relativ einfach über WMI. Dazu fragen Sie über eine Instanz der Klasse `System.Management.ManagementObject` ein Objekt der WMI-Klasse `Win32_OperatingSystem` ab. Über verschiedene Eigenschaften dieser Klasse erhalten Sie (u. a.) Informationen zum Arbeitsspeicher (in KB).

Die Methode `GetRamInformation` in Listing 324 liest diese Informationen aus dem lokalen System in eine Instanz der Klasse `RamInformation` aus und gibt eine Referenz darauf zurück. `GetRamInformation` erzeugt dazu eine Auflistung der WMI-Objekte, die die auf einem System installierten Betriebssysteme repräsentieren, und fragt das erste dieser Objekte ab (das das aktuelle System repräsentiert).

Zum Kompilieren dieses Quellcodes benötigt das Programm eine Referenz auf die Assembly *System.Management.dll*. Außerdem müssen Sie den Namensraum `System.Management` einbinden. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
/* Verwaltet Arbeitsspeicher-Informationen */
public class RamInformation
{
   /* Der freie physische Speicher in KB */
   public ulong FreePhysicalMemory = 0;

   /* Der freie Speicher in den Auslagerungsdateien in KB */
   public ulong FreeSpaceInPagingFiles = 0;

   /* Der freie virtuelle Speicher in KB.
      In der Regel handelt es sich dabei um den freien physischen Speicher
      addiert mit dem freien Speicher in den Auslagerungsdateien. */
   public ulong FreeVirtualMemory = 0;

   /* Maximale Anzahl an KB, die einem Prozess zugeordnet werden können */
   public ulong MaxProcessMemorySize = 0;

   /* Der gesamte virtuelle Speicher in KB */
   public ulong TotalVirtualMemorySize = 0;

   /* Der Arbeitsspeicher in KB, der dem Betriebssystem als
      verfügbar gemeldet wurde */
   public ulong TotalVisibleMemorySize = 0;
}

/* Ermittelt Arbeitsspeicher-Informationen */
public static RamInformation GetRamInformation()
{
   // Instanz der Klasse RamInformation erzeugen
   RamInformation ri = new RamInformation();

   // Speicher-Infos über WMI abfragen
   ManagementClass mc = new ManagementClass("Win32_OperatingSystem");
   ManagementObjectCollection moc = mc.GetInstances();
   ManagementObjectCollection.ManagementObjectEnumerator
      mocEnumerator = moc.GetEnumerator();
   mocEnumerator.MoveNext();
   ManagementObject mo = (ManagementObject)mocEnumerator.Current;

   try
   {
      ri.FreePhysicalMemory = (ulong)mo["FreePhysicalMemory"];
   }
   catch {}

   try
```

Listing 324: Methode zur Ermittlung von Informationen zum Arbeitsspeicher

```
    {
        ri.FreeSpaceInPagingFiles = (ulong)mo["FreeSpaceInPagingFiles"];
    }
    catch {}

    try
    {
        ri.FreeVirtualMemory = (ulong)mo["FreeVirtualMemory"];
    }
    catch {}

    try
    {
        ri.MaxProcessMemorySize = (ulong)mo["MaxProcessMemorySize"];
    }
    catch {}

    try
    {
        ri.TotalVirtualMemorySize = (ulong)mo["TotalVirtualMemorySize"];
    }
    catch {}

    try
    {
        ri.TotalVisibleMemorySize = (ulong)mo["TotalVisibleMemorySize"];
    }
    catch {}

    // Speicher der WMI-Objekte freigeben um den Arbeitsspeicher
    // möglichst schnell zu entlasten
    mo.Dispose();
    moc.Dispose();
    mc.Dispose();

    // Informationen zurückgeben
    return ri;
}
```

Listing 324: Methode zur Ermittlung von Informationen zum Arbeitsspeicher (Forts.)

Voraussetzung für die fehlerfreie Ausführung dieser Methode ist, dass auf dem Zielsystem WMI installiert ist und dass das Benutzerkonto, das aktuell verwendet wird, über ausreichende WMI- und Systemrechte verfügt. Falls das interaktive Konto nicht über ausreichend Rechte verfügt, können Sie vor der Ausführung dieser Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 307 beschreibe.

Listing 325 zeigt eine Anwendung dieser Methode, bei der die ermittelten Daten an der Konsole ausgegeben werden.

```
SystemUtils.RamInformation ri = SystemUtils.GetRamInformation();

Console.WriteLine("FreePhysicalMemory: {0} KB",
   ri.FreePhysicalMemory);
Console.WriteLine("FreeSpaceInPagingFiles: {0} KB",
   ri.FreeSpaceInPagingFiles);
Console.WriteLine("FreeVirtualMemory: {0} KB",
   ri.FreeVirtualMemory);
Console.WriteLine("MaxProcessMemorySize: {0} KB",
   ri.MaxProcessMemorySize);
Console.WriteLine("TotalVirtualMemorySize: {0} KB",
   ri.TotalVirtualMemorySize);
Console.WriteLine("TotalVisibleMemorySize: {0} KB",
   ri.TotalVisibleMemorySize);
```


Listing 325: Anwendung der Methode zur Ermittlung von Informationen zum Arbeitsspeicher

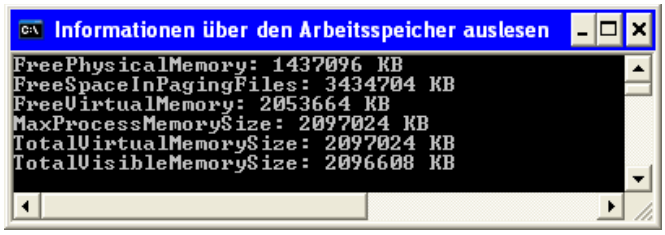


Abbildung 105: Das Beispielprogramm zum Auslesen von Arbeitsspeicher-Informationen in Aktion

193 Die Version des aktuellen Service-Packs ermitteln

Die Version des aktuellen Service-Packs können Sie über die Eigenschaft `ServicePack` von `Environment.OSVersion` ermitteln. Leider liefert diese Eigenschaft nur einen String zurück, der die Version in einer verkürzten Schreibweise enthält. Die Version 2.0 wird zum Beispiel als »2« zurückgegeben. Wenn Sie in Ihrem Programm überprüfen müssen, ob ein Service-Pack ab einer bestimmten Version (z. B. ab Version 1.1) installiert ist, haben Sie damit unnötigen Aufwand. Außerdem können Sie nie genau wissen, in welcher Form die aktuelle Version (in Zukunft) zurückgegeben wird. Deshalb verzichte ich auf diese Eigenschaft.

Informationen zum aktuell installierten Service-Pack können Sie auch über die WMI-Klasse `Win32_OperatingSystem` auslesen. Die Eigenschaften `ServicePackMajorVersion` und `ServicePackMinorVersion`, die beides `ushort`-Werte sind, liefern Ihnen genaue Informationen über die Version.

Die Methode `GetServicePackVersion` in Listing 326 ermittelt eine Instanz der Klasse `Win32_OperatingSystem` für das aktive Betriebssystem, liest diese Eigenschaften aus und gibt eine Instanz der Klasse `OSVersion` mit Informationen zu der Version zurück. Ich verzichte auf eine nähere Beschreibung dieser Methode, da die Vorgehensweise sich prinzipiell nicht von der des Rezepts 192 unterscheidet.

Zum Kompilieren der Methode müssen Sie die Assembly *System.Management.dll* referenzieren und die Namensräume `System` und `System.Management` einbinden. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
/* Verwaltet die Version des Service-Pack */
public class ServicePackVersion
{
   /* Die Hauptversion */
   public ushort Major;

   /* Die Nebenversion */
   public ushort Minor;
}

/* Ermittelt die Version des aktuellen Service-Pack */
public static ServicePackVersion GetServicePackVersion()
{
   // Version-Instanz für die Rückgabe erzeugen
   ServicePackVersion version = new ServicePackVersion();

   // Win32_OperatingSystem-Instanz für das aktive Betriebssystem ermitteln
   ManagementClass mc = new ManagementClass("Win32_OperatingSystem");
   ManagementObjectCollection moc = mc.GetInstances();
   ManagementObjectCollection.ManagementObjectEnumerator
      mocEnumerator = moc.GetEnumerator();
   mocEnumerator.MoveNext();
   ManagementObject mo = (ManagementObject)mocEnumerator.Current;

   // Die Version abfragen und speichern
   try
   {
      version.Major =(ushort)mo["ServicePackMajorVersion"];
   }
   catch {}
   try
   {
      version.Minor =(ushort)mo["ServicePackMinorVersion"];
   }
   catch {}

   // WMI-Objekte freigeben um den Speicher möglichst schnell zu entlasten
   mo.Dispose();
   moc.Dispose();
   mc.Dispose();
```

Listing 326: Methode zur Ermittlung der Version des aktuellen Service-Packs

```
    // Ergebnis zurückgeben
    return version;
}
```

Listing 326: Methode zur Ermittlung der Version des aktuellen Service-Packs (Forts.)

Wie bereits in den vorhergehenden Rezepten benötigt das Programm eine Referenz auf die Assembly *System.Management.dll*. Damit es fehlerfrei ausgeführt werden kann, muss auf dem Zielsystem WMI installiert sein. Außerdem muss das Benutzerkonto, das aktuell verwendet wird, über ausreichend WMI- und Systemrechte verfügen. Falls das aktuelle Benutzerkonto nicht über ausreichend Rechte verfügt, können Sie vor der Ausführung der `GetServicePackVersion`-Methode das Programm mit einem anderen Benutzerkonto personifizieren, wie ich es im Rezept 307 beschreibe.

194 Installierte Programme auflisten

Das Auflisten aller Programme, die auf einem System installiert sind, ist leider nicht allzu einfach. Über die WMI-Klasse `Win32_Product` erhalten Sie lediglich Informationen zu Programmen, die über den Windows-Installer installiert wurden. Viele Installations-Tools arbeiten aber unabhängig vom Windows-Installer. Die mit diesen Tools installierten Anwendungen werden von der Klasse `Win32_Product` nicht berücksichtigt.

Natürlich können Sie nicht jedes installierte Programm finden. Problematisch werden z. B. .NET-Programme, die durch einfaches Kopieren auf ein System übertragen wurden. Sinnvoll wäre aber wenigstens das Auflisten der Programme, die in Windows als deinstallierbar registriert sind und in der Software-Liste der Systemsteuerung auftauchen.

Um diese Programme zu ermitteln, können Sie die Registry abfragen. Die deinstallierbaren Programme finden Sie im Schlüssel *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows-CurrentVersion\Uninstall*. Zum Zugriff auf die Registry können Sie die WMI-Klasse `StdRegProv` aus dem Namensraum *root\default* verwenden. Die Verwendung dieser Klasse würde den Vorteil bringen, auch die auf entfernten Systemen installierten Programme auflisten zu können. Abgesehen davon, dass die `StdRegProv`-Klasse in Windows 95, 98 und Me (laut der Dokumentation) nicht zur Verfügung steht, ist deren Anwendung auch nicht allzu einfach. Problematisch ist z. B. die (mir nicht gelungene ☺) By-Reference-Übergabe eines Arrays als COM-Variant-Typ beim Aufruf der WMI-Methode `EnumKey`. Also habe ich eine auf dem lokalen System funktionierende Lösung über den direkten Zugriff auf die Registry über eine `RegistryKey`-Instanz implementiert.

Die Methode `EnumInstalledPrograms` in Listing 327 erzeugt deshalb ein solches Objekt für den Schlüssel, der die Informationen zu der deinstallierbaren Software enthält. Über die Methode `GetSubKeyNames` ermittelt `EnumInstalledPrograms` die Namen der in diesem Schlüssel enthaltenen Unterschlüssel, geht diese durch und öffnet jeden Unterschlüssel über `OpenSubKey`. Um den Namen des Programms zu ermitteln, liest `EnumInstalledPrograms` zunächst den Wert `DisplayName` aus, der bei echten Programmen in der Regel den anzuzeigenden Programmnamen enthält (bei installierten Komponenten, die keine Programme darstellen, existiert dieser Wert nicht). Ist `DisplayName` nicht vorhanden, versucht `EnumInstalledPrograms` noch den eventuell vorhandenen Wert `QuietDisplayName` auszulesen. Kann aus einem dieser Werte ein Name aus-

gelesen werden, wird das als installiertes Programm gewertet und der Name der Ergebnis-Auflistung hinzugefügt.

Die Methode erfordert die Referenzierung der Assembly *System.Management.dll* und die Einbindung der Namensräume `System`, `System.Management`, `System.Collections.Generic`, `System.Collections.ObjectModel` und `Microsoft.Win32`. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
public static ReadOnlyCollection<string> EnumInstalledPrograms()
{
   // Auflistung für die Rückgabe erzeugen
   List<string> installedPrograms = new List<string>();

   // RegistryKey-Instanz für den Schlüssel HKEY_LOCAL_MACHINE\Software\
   // Microsoft\Windows\CurrentVersion\Uninstall erzeugen
   string keyPath = @"Software\Microsoft\Windows\CurrentVersion\Uninstall";
   RegistryKey regKey = Registry.LocalMachine.OpenSubKey(keyPath);
   if (regKey != null)
   {
      // Alle Unterschlüssel durchgehen
      string[] subKeyNames = regKey.GetSubKeyNames();
      for (int i = 0; i < subKeyNames.Length; i++)
      {
         // Unterschlüssel öffnen und den Wert DisplayName oder
         // QuietDisplayName auslesen
         RegistryKey subKey = regKey.OpenSubKey(subKeyNames[i]);
         string programName = (string)subKey.GetValue("DisplayName");
         if (programName == null)
         {
            programName = (string)subKey.GetValue("QuietDisplayName");
         }
-*
         // Wenn ein Programmname ermittelt werden konnte, diesen an die
         // String-Auflistung anfügen
         if (programName != null)
         {
            installedPrograms.Add(programName);
         }
      }
   }
   else
   {
      // Schlüssel nicht gefunden: Ausnahme generieren
      throw new Exception("Registry-Schlüssel " +
```

Listing 327: Methode zur Auflistung der installierten (und deinstallierbaren) Anwendungen

```
            Registry.LocalMachine.Name + "\\" + keyPath + " nicht gefunden");
      }

      // Ergebnis-Liste zurückgeben
      return new ReadOnlyCollection<string>(installedPrograms);
   }
```

Listing 327: Methode zur Auflistung der installierten (und deinstallierbaren) Anwendungen (Forts.)

Das folgende Listing zeigt eine Beispielanwendung:

```
try
{
   StringCollection installedPrograms = EnumInstalledPrograms();
   foreach (string programName in installedPrograms)
   {
      Console.WriteLine(programName);
   }
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 328: Anwendung der Methode zur Auflistung der installierten Anwendungen

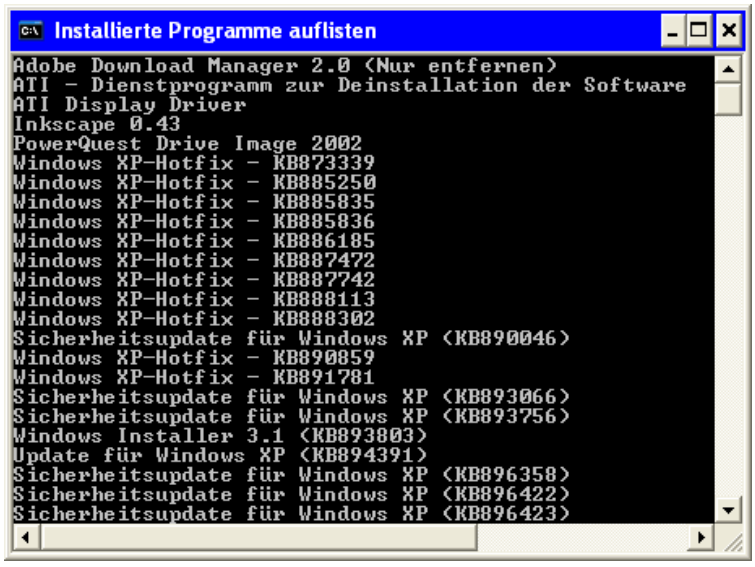


Abbildung 106: Auflisten der installierten Programme

195 Informationen zu den parallelen Schnittstellen des Systems auslesen

Informationen über die parallelen Schnittstellen eines Systems erhalten Sie über die WMI-Klasse `Win32_ParallelPort`. Die Methode `EnumParallelPorts` in Listing 330 ermittelt über diese Klasse die wichtigsten Informationen zu den parallelen Schnittstellen des lokalen Computers.

Um diese speichern zu können, sind zunächst zwei Klassen und eine Aufzählung implementiert. Die Klasse `ParallellPortCapabilities` verwaltet die Eigenschaften einer parallelen Schnittstelle und wird als Typ der Eigenschaft `Capabilities` in der Klasse `ParallelPort` eingesetzt. Die Aufzählung `ParallellPortStatusInfo` wird zur Speicherung des Status eines Ports verwendet. Instanzen der Klasse `ParallelPort` speichern dann später die wichtigsten Daten einer parallelen Schnittstelle.

Zum Kompilieren dieses Quellcodes müssen Sie die Assembly *System.Management.dll* referenzieren und die Namensräume `System`, `System.Management` und `System.Collections.Generic` einbinden.


```
/* Verwaltet die Fähigkeiten eines parallelen Ports */
public class ParallellPortCapabilities
{
   /* XT/AT-kompatibel */
   public bool XTATCompatible;

   /* PS2-kompatibel */
   public bool PS2Compatible;

   /* ECP */
   public bool ECP;

   /* EPP */
   public bool EPP;

   /* PC98 */
   public bool PC98;

   /* PC98Hireso */
   public bool PC98Hireso;

   /* PCH98 */
   public bool PCH98;

   /* Der Port besitzt weitere Fähigkeiten */
   public bool Other;
}
```

Listing 329: Klassen und eine Aufzählung zur Speicherung der wichtigsten Eigenschaften eines parallelen Ports

```
/* Aufzählung für den Status eines parallelen Ports */
public enum ParallelPortStatusInfo: ushort
{
   /* Der Status ist nicht definiert */
   NotDefined = 0,

   /* Der Port besitzt einen anderen, hier nicht genannten Status */
   Other = 1,

   /* Der Status ist unbekannt */
   Unknown = 2,

   /* Der Port ist aktiviert */
   Enabled = 3,

   /* Der Port ist deaktiviert */
   Disabled = 4,

   /* Der Status ist zurzeit nicht anwendbar */
   NotApplicable = 5
}

/* Klasse mit den wichtigsten Eigenschaften eines parallelen Ports */
public class ParallelPort
{
   /* Die Geräte-Id des Ports */
   public string DeviceId;

   /* Die Fähigkeiten des Ports */
   public ParallellPortCapabilities Capabilities =
      new ParallellPortCapabilities();

   /* Gibt an, ob der Port DMA unterstützt */
   public bool DMASupport;

   /* Gibt den Status des Ports an */
   public ParallelPortStatusInfo StatusInfo;
}
```

Listing 329: Klassen und eine Aufzählung zur Speicherung der wichtigsten Eigenschaften eines parallelen Ports (Forts.)

Leider sind die einzelnen Werte für die Fähigkeiten und den Status des Ports in der WMI-Dokumentation nicht beschrieben, weswegen ich für mir unbekannte Werte (wie zum Beispiel EPP und ECP) leider keine Beschreibung anfügen konnte.

Die Methode `EnumParallelPorts` erzeugt für die WMI-Klasse `Win32_ParallelPort` eine Auflistung, geht diese durch und speichert die eingelesenen Daten in Instanzen der Klasse `Parallel-`

Port. Das Einlesen erfolgt in einer Ausnahmebehandlung um Fehler bei Eigenschaften zu vermeiden, die null speichern. Das Feld StatusInfo, das ja eine Aufzählung vom Typ ParallelPortStatusInfo ist, kann direkt konvertiert werden, da die Werte dieser Aufzählung mit den Statusinfo-Werten korrespondieren.

Das Auslesen der Fähigkeiten eines Ports ist etwas komplexer, da diese in einem uint-Array verwaltet werden. EnumParallelPorts geht dieses Array durch, vergleicht die gespeicherten Werte mit den dokumentierten Konstanten und belegt die entsprechende Eigenschaft der Capabilities-Eigenschaft des ParallelPort-Objekts mit true, wenn eine Fähigkeit gesetzt ist.

Die so definierten Ports werden dann der anfangs erzeugten List<ParallelPort>-Auflistung hinzugefügt, die am Ende der Methode als ReadOnlyCollection<ParallelPort> zurückgegeben wird.


```
public static ReadOnlyCollection<ParallelPort> EnumParallelPorts()
{
   // List-Instanz für die Rückgabe erzeugen
   List<ParallelPort> ports = new List<ParallelPort>();

   // WMI-Auflistung der Objekte der Win32_ParallelPort-Klasse erzeugen
   ManagementClass mc = new ManagementClass("Win32_ParallelPort");
   ManagementObjectCollection moc = mc.GetInstances();

   // Die einzelnen Objekte durchgehen
   foreach (ManagementBaseObject mbo in moc)
   {
      // ParallelPort-Instanz erzeugen und initialisieren
      ParallelPort port = new ParallelPort();
      try
      {
         port.DeviceId = (string)mbo["DeviceId"];
      }
      catch { }

      try
      {
         port.DMASupport = (bool)mbo["DMASupport"];
      }
      catch { }

      try
      {
         port.StatusInfo = (ParallelPortStatusInfo)mbo["StatusInfo"];
      }
      catch { }
```

Listing 330: Methode zum Auflisten der parallelen Ports des Systems

```
    // Fähigkeiten aus dem Capabilities-Array auslesen
    try
    {
       uint[] caps = (uint[])mbo["Capabilities"];
       for (int i = 0; i < caps.Length; i++)
       {
          switch (caps[i])
          {
             case 2:  // Other
                port.Capabilities.Other = true;
                break;
             case 3:  // XT/AT Compatible
                port.Capabilities.XTATCompatible = true;
                break;
             case 4:  //  PS/2 Compatible
                port.Capabilities.PS2Compatible = true;
                break;
             case 5:  //  ECP
                port.Capabilities.ECP = true;
                break;
             case 6:  //  EPP
                port.Capabilities.EPP = true;
                break;
             case 7:  //  PC-98
                port.Capabilities.PC98 = true;
                break;
             case 8:  //  PC-98-Hireso
                port.Capabilities.PC98Hireso = true;
                break;
             case 9:  //  PC-H98
                port.Capabilities.PCH98 = true;
                break;
          }
       }
    }
    catch { }

    // ParallelPort-Instanz an die Auflistung anfügen
    ports.Add(port);
}

// Speicher der WMI-Objekte freigeben um den Arbeitsspeicher möglichst
// schnell zu entlasten
moc.Dispose();
mc.Dispose();
```

Listing 330: Methode zum Auflisten der parallelen Ports des Systems (Forts.)

```
   // Ergebnis zurückgeben
   return new ReadOnlyCollection<ParallelPort>(ports);
}
```

Listing 330: Methode zum Auflisten der parallelen Ports des Systems (Forts.)

Hinweis

Die Abfrage der parallelen Ports ergab auf meinem System (Windows XP Professional) für den Status, den (vorhandenen) DMA-Support und die Fähigkeiten immer den Wert `null`. Im Internet habe ich dazu leider nichts gefunden. Na ja, wenigstens die Eigenschaft `DeviceId` war immer vorhanden ...

Das folgende Listing zeigt eine Anwendung der Methode `EnumParallelPorts`:


```
/* Auflisten der parallelen Ports des Systems */
foreach(var port in EnumParallelPorts())
{
   Console.WriteLine("DeviceId: {0}", port.DeviceId);
   Console.WriteLine("DMASupport: {0}", port.DMASupport);
   Console.WriteLine("StatusInfo: {0}", port.StatusInfo);
   Console.WriteLine("Capabilities:");
   Console.WriteLine("ECP: {0}", port.Capabilities.ECP);
   Console.WriteLine("EPP: {0}", port.Capabilities.EPP);
   Console.WriteLine("PC98: {0}", port.Capabilities.PC98);
   Console.WriteLine("PC98Hireso: {0}", port.Capabilities.PC98Hireso);
   Console.WriteLine("PCH98: {0}", port.Capabilities.PCH98);
   Console.WriteLine("PS2Compatible: {0}", port.Capabilities.PS2Compatible);
   Console.WriteLine("Other: {0}", port.Capabilities.Other);
   Console.WriteLine();
}
```

Listing 331: Beispielhafte Anwendung der Methode zum Auflisten der parallelen Ports des Systems

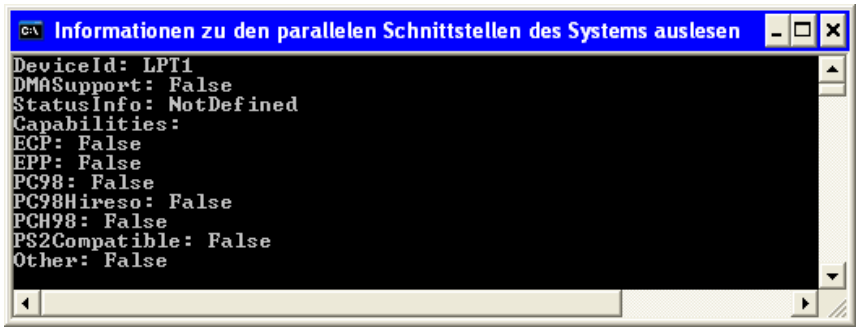


Abbildung 107: Das Beispielprogramm zur Ermittlung der wichtigsten Informationen zu den parallelen Schnittstellen des Systems

196 Informationen zu den seriellen Schnittstellen des Systems auslesen

Ähnlich dem Rezept 195 können Sie über die WMI-Klasse `Win32_SerialPort` (eingeschränkte) Informationen zu den seriellen Schnittstellen des lokalen Systems auslesen. Die Methode `EnumSerialPorts` Listing 335 arbeitet deswegen auf nahezu dieselbe Weise wie die Methode `EnumParallelPorts` aus dem genannten Rezept.

`EnumSerialPorts` gibt eine Auflistung des Typs `SerialPorts` mit Instanzen der Klasse `SerialPort` zurück. Die Klasse `SerialPort` enthält die wichtigsten Eigenschaften einer seriellen Schnittstelle.

`EnumSerialPorts` benötigt eine Referenz auf die Assembly *System.Management.dll* und den Import der Namensräume `System`, `System.Management`, `System.Collections.Generic` und `System.Collections.ObjectModel`.Auf dem ausführenden Rechner muss WMI installiert sein.

Zunächst habe ich eine Klasse implementiert, die die Fähigkeiten eines seriellen Ports verwaltet. Die einzelnen Fähigkeiten sind in der WMI-Dokumentation leider nicht dokumentiert.

```
public class SerialPortCapabilities
{
   /* Unbekannte Fähigkeit */
   public bool Unknown;

   /* XT/AT-kompatibel */
   public bool CompatibleXTAT;

   /* 16450-kompatibel */
   public bool Compatible16450;

   /* 16550-kompatibel */
   public bool Compatible16550;

   /* 16550A-kompatibel */
   public bool Compatible16550A;
   /* 8251-kompatibel */
   public bool Compatible8251;

   /* 8251FIFO-kompatibel */
   public bool Compatible8251FIFO;

   /* Andere Fähigkeit, die hier nicht aufgelistet wird */
   public bool Other;
}
```

Listing 332: Klasse zur Verwaltung der Fähigkeiten eines seriellen Ports

Dann folgt eine Aufzählung, die den Status (der leider auch wieder nicht dokumentiert ist) des seriellen Ports angibt:

```
public enum SerialPortStatusInfo : ushort
{
   /* Der Status ist nicht definiert */
   NotDefined = 0,

   /* Der Port besitzt einen anderen, hier nicht genannten Status */
   Other = 1,

   /* Der Status ist unbekannt */
   Unknown = 2,

   /* Der Port ist aktiviert */
   Enabled = 3,

   /* Der Port ist deaktiviert */
   Disabled = 4,

   /* Der Status ist zurzeit nicht anwendbar */
   NotApplicable = 5
}
```

Listing 333: Aufzählung für den Status eines seriellen Ports

Die Klasse `SerialPort` verwaltet die Eigenschaften eines seriellen Ports:

```
/* Verwaltet die wichtigsten Eigenschaften eines seriellen Ports */
public class SerialPort
{
   /* Die Geräte-ID */
   public string DeviceId;

   /* Die maximale Baudrate */
   public uint MaxBaudRate;

   /* Gibt an, ob der Port den 16-Bit-Modus unterstützt */
   public bool Supports16BitMode;

   /* Gibt an, ob der Port DTR- und DSR-Signale unterstützt.
      Diese Signale werden üblicherweise für einen Handshake verwendet. */
   public bool SupportsDTRDSR;

   /* Gibt an, ob der Port Laufzeit-Timeouts unterstützt.
      Laufzeit-Timeouts beziehen sich auf die gesamte Zeit,
      die zwischen Datenübertragungen vergeht */
   public bool SupportsElapsedTimeouts;
```

Listing 334: Klasse zur Verwaltung der Eigenschaften eines seriellen Ports

```
   /* Gibt an, ob der Port Intervall-Timeouts unterstützt.
      Intervall-Timeouts beziehen sich auf die Zeit, die zwischen
      dem Empfang von einzelnen Datenpaketen vergeht. */
   public bool SupportsIntervallTimeouts;

   /* Gibt an, ob der Port eine Paritäts-Überprüfung unterstützt. */
   public bool SupportsParityCheck;

   /* Gibt an, ob der Port RLSD (Received Line Signal Detect) unterstützt */
   public bool SupportsRLSD;

   /* Gibt an, ob der Port RTS- und CTS-Signale unterstützt
      Diese Signale werden üblicherweise für einen Handshake verwendet. */
   public bool SupportsRTSCTS;

   /* Gibt an, ob der Port spezielle Zeichen unterstützt */
   public bool SupportsSpecialCharacters;

   /* Gibt an, ob der Port eine XOn/XOff-Flusskontrolle unterstützt */
   public bool SupportsXOnXOff;

   /* Gibt an, ob der Port die Einstellung der
      XOn/XOff-Flusskontrolle unterstützt */
   public bool SupportsXOnXOffSettings;

   /* Die Fähigkeiten des Ports */
   public SerialPortCapabilities Capabilities =
      new SerialPortCapabilities();

   /* Information über den Status des Ports */
   public SerialPortStatusInfo StatusInfo;
}
```

Listing 334: Klasse zur Verwaltung der Eigenschaften eines seriellen Ports (Forts.)

Schließlich folgt noch die Methode, die die Informationen über die seriellen Schnittstellen des Systems über WMI ausliest, in einzelne `SerialPort`-Objekt schreibt und diese als `ReadOnlyCollection<SerialPort>`-Auflistung zurückgibt:

```
public static ReadOnlyCollection<SerialPort> EnumSerialPorts()
{
   // List-Instanz für die Rückgabe erzeugen
   List<SerialPort> ports = new List<SerialPort>();

   // WMI-Auflistung der Objekte der Win32_SerialPort erzeugen
   ManagementClass mc = new ManagementClass("Win32_SerialPort");
```

Listing 335: Methode zum Auflisten der seriellen Schnittstellen eines Systems

```
ManagementObjectCollection moc = mc.GetInstances();

// Die einzelnen Objekte durchgehen
foreach (ManagementBaseObject mbo in moc)
{
   // SerialPort-Instanz erzeugen und initialisieren
   SerialPort port = new SerialPort();
   try
   {
      port.DeviceId = (string)mbo["DeviceId"];
   }
   catch { }

   try
   {
      port.MaxBaudRate = (uint)mbo["MaxBaudRate"];
   }
   catch { }

   try
   {
      port.Supports16BitMode = (bool)mbo["Supports16BitMode"];
   }
   catch { }

   try
   {
      port.SupportsDTRDSR = (bool)mbo["SupportsDTRDSR"];
   }
   catch { }

   try
   {
      port.SupportsElapsedTimeouts =
         (bool)mbo["SupportsElapsedTimeouts"];
   }
   catch { }

   try
   {
      port.SupportsIntervallTimeouts = (bool)mbo["SupportsIntTimeouts"];
   }
   catch { }

   try
```

Listing 335: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

```
{
   port.SupportsParityCheck = (bool)mbo["SupportsParityCheck"];
}
catch { }

try
{
   port.SupportsRLSD = (bool)mbo["SupportsRLSD"];
}
catch { }

try
{
   port.SupportsRTSCTS = (bool)mbo["SupportsRTSCTS"];
}
catch { }

try
{
   port.SupportsSpecialCharacters =
      (bool)mbo["SupportsSpecialCharacters"];
}
catch { }

try
{
   port.SupportsXOnXOff = (bool)mbo["SupportsXOnXOff"];
}
catch { }

try
{
   port.SupportsXOnXOffSettings = (bool)mbo["SupportsXOnXOffSet"];
}
catch { }

try
{
   port.StatusInfo = (SerialPortStatusInfo)mbo["StatusInfo"];
}
catch { }

// Fähigkeiten aus dem Capabilities-Array auslesen
try
{
   uint[] caps = (uint[])mbo["Capabilities"];
```

Listing 335: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

```
            for (int i = 0; i < caps.Length; i++)
            {
               switch (caps[i])
               {
                  case 1:    // Other
                     port.Capabilities.Other = true;
                     break;
                  case 2:    // Unknown
                     port.Capabilities.Unknown = true;
                     break;
                  case 3:    // XT/AT Compatible
                     port.Capabilities.CompatibleXTAT = true;
                     break;
                  case 4:    // 16450 Compatible
                     port.Capabilities.Compatible16450 = true;
                     break;
                  case 5:    // 16550 Compatible
                     port.Capabilities.Compatible16550 = true;
                     break;
                  case 6:    // 16550A Compatible
                     port.Capabilities.Compatible16550A = true;
                     break;
                  case 160:  // 8251 Compatible
                     port.Capabilities.Compatible8251 = true;
                     break;
                  case 161:  // 8251FIFO Compatible
                     port.Capabilities.Compatible8251FIFO = true;
                     break;
               }
            }
         }
         catch { }

         // SerialPort-Instanz an die Auflistung anfügen
         ports.Add(port);
      }

      // Speicher des WMI-Objekts freigeben um den Arbeitsspeicher möglichst
      // schnell zu entlasten
      moc.Dispose();
      mc.Dispose();

      // Ergebnis zurückgeben
      return new ReadOnlyCollection<SerialPort>(ports);
   }
```

Listing 335: Methode zum Auflisten der seriellen Schnittstellen eines Systems (Forts.)

Hinweis

Ähnlich der Abfrage der parallelen Ports ergaben die Fähigkeiten (`Capabilities`) auf meinem System (Windows XP Professional) immer den Wert `null`. Anders als bei der Klasse `Win32_ParallelPort` war aber die Eigenschaft `StatusInfo` verfügbar.

197 Ermitteln der Soundkarten des Systems

Über die WMI-Klasse `Win32_SoundDevice` können Sie Informationen zu den installierten Soundkarten auslesen. Leider sind die meisten (von der Basisklasse `CIM_LogicalDevice` geerbten) Eigenschaften dieser Klasse scheinbar nicht implementiert und geben `null` zurück. Aber eigentlich interessiert uns ja nur, ob überhaupt eine Soundkarte installiert ist und wenn ja, welche.

Die Methode `EnumSoundcards` in Listing 336 liest deshalb nur die Geräte-Id und den Produktnamen der installierten Soundkarten in eine Instanz der generischen Klasse `Dictionary`, deren Schlüssel und Wert jeweils mit dem Typ `string` typisiert ist. Ich beschreibe die Funktionsweise dieser Methode nicht näher, da sie im Prinzip nicht anders vorgeht als die Methoden in den vorhergehenden Rezepten.

Zum Kompilieren dieser Methode benötigen Sie eine Referenz auf die Assembly *System.Management.dll*. Außerdem müssen Sie die Namensräume `System`, `System.Management` und `System.Collections.Generic` importieren.

```
public static Dictionary<string, string> EnumSoundcards()
{
   // HashTable für die Rückgabe erzeugen
   Dictionary<string, string> soundcards = new Dictionary<string, string>();

   // Abfragen der installierten Soundkarten
   ManagementObjectSearcher searcher = new ManagementObjectSearcher(
      "SELECT * FROM Win32_SoundDevice");
   ManagementObjectCollection moc = searcher.Get();
   foreach (ManagementBaseObject mbo in moc)
   {
      // DeviceId (als Schlüssel) und ProductName (als Wert)
      // an die Hashtable-Instanz anfügen
      soundcards.Add(
         mbo["DeviceId"] != null ?
            mbo["DeviceId"].ToString() : String.Empty,
         mbo["ProductName"] != null ?
            mbo["ProductName"].ToString() : String.Empty);
   }
```

Listing 336: Methode zum Auflisten der wichtigsten Informationen zu den installierten Soundkarten

```
   // ManagementObjectSearcher freigeben, um den Speicher möglichst schnell
   // zu entlasten
   searcher.Dispose();

   // StringCollection zurückgeben
   return soundcards;
}
```

Listing 336: Methode zum Auflisten der wichtigsten Informationen zu den installierten Soundkarten (Forts.)

Wenn Sie nun lediglich ermitteln wollen, ob eine Soundkarte installiert ist, können Sie das folgende Programm einsetzen:

```
Dictionary<string, string> soundcards = EnumSoundcards();
if (soundcards.Count = 0)
{
   MessageBox.Show("Auf Ihrem System wurde keine Soundkarte gefunden",
      Application.ProductName, MessageBoxButtons.OK,
      MessageBoxIcon.Exclamation);
}
```

Listing 337: Einfache Abfrage danach, ob auf dem System eine Soundkarte installiert ist

198 Windows herunterfahren und neu starten

Über die Methoden `Reboot` und `Shutdown` einer Instanz der WMI-Klasse `Win32_OperatingSystem` können Sie ein System neu starten oder herunterfahren. Alternativ könnten Sie dazu auch die API-Funktion `ExitWindowsEx` einsetzen. Das dabei notwendige Erlangen des Reboot-Privilegs ist aber recht kompliziert.

Die Methode `ShutdownSystem` in Listing 338 besitzt die Argumente `hostName`, `remoteAuthUser` und `remoteAuthPassword`, um auch das Herunterfahren und Booten eines entfernten Systems zu ermöglichen. Mit diesen Informationen erzeugt die Methode zunächst ein `ManagementScope`-Objekt. Da das Herunterfahren und Neu-Starten des Systems das Privileg `SE_SHUTDOWN_NAME` (Herunterfahren des Systems) erfordert, setzt `ShutdownSystem` zusätzlich die Eigenschaft `EnablePrivileges` dieses Objekts auf `true`. Damit wird laut der Dokumentation erreicht, dass die Privilegien des aktuellen Benutzerkontos für WMI aktiviert werden und somit zur Verfügung stehen. Für das Herunterfahren und Neu-Booten eines entfernten Systems ist das Setzen dieser Eigenschaft allerdings nicht notwendig, wenn Sie das `ManagementScope`-Objekt mit Authentifizierungsinformationen versehen.

Das `ManagementScope`-Objekt wird bei der Erzeugung einer `ManagementClass`-Instanz übergeben, womit die dort eingestellte Konfiguration beim folgenden Erzeugen der `ManagementObjectCollection`-Auflistung für die verfügbaren Betriebssysteme übernommen wird. Über den Enumerator dieser Auflistung ermittelt `ShutdownSystem` dann eine Referenz auf ein `ManagementObject`-Objekt des aktiven Betriebssystems und ruft über die `Invoke`-Methode die `Reboot`- oder `Shutdown`-Methode des `Win32_OperatingSystem`-Objekts auf.

Die Methode `ShutdownSystem` benötigt eine Referenz auf die Assembly *System.Management.dll* und den Import des Namensraums `System.Management`. Auf dem Zielcomputer muss zur Ausführung des Programms WMI installiert sein.

```
public static void ShutdownSystem(string hostName, string remoteAuthUser,
   string remoteAuthPassword, bool reboot)
{
   // Den Hostnamen auf localhost setzen, wenn keiner übergeben wurde
   if (hostName == null || hostName == "")
   {
      hostName = "localhost";
   }

   // ManagementScope-Objekt erzeugen, damit die EnablePrivileges-Eigenschaft
   // gesetzt und die Authentifizierungs-Informationen übergeben werden
   // können
   ManagementScope scope = new ManagementScope(@"\\" + hostName +
      "\\root\\cimv2");
   scope.Options.EnablePrivileges = true;
   scope.Options.Username = remoteAuthUser;
   scope.Options.Password = remoteAuthPassword;

   // Win32_OperatingSystem-Instanz für das aktive Betriebssystem ermitteln
   ManagementClass mc = new ManagementClass(scope, new ManagementPath(
      "Win32_OperatingSystem"), new ObjectGetOptions());
   ManagementObjectCollection moc = mc.GetInstances();
   ManagementObjectCollection.ManagementObjectEnumerator
      mocEnumerator = moc.GetEnumerator();
   mocEnumerator.MoveNext();
   ManagementObject mo = (ManagementObject)mocEnumerator.Current;

   // Rechner über die Win32Shutdown-Methode neu starten oder herunterfahren
   if (reboot)
   {
      mo.InvokeMethod("Reboot", null);
   }
   else
   {
      mo.InvokeMethod("Shutdown", null);
   }

   // WMI-Objekte freigeben um den Speicher möglichst schnell zu entlasten
   mo.Dispose();
   moc.Dispose();
   mc.Dispose();
}
```

Listing 338: Methode zum Herunterfahren oder Rebooten des Systems

Zur erfolgreichen Ausführung der Methode `ShutdownSystem` muss natürlich zunächst WMI auf dem Zielsystem installiert sein. Zudem muss das Benutzerkonto, das für die WMI-Authentifizierung verwendet wird, über das Privileg `SE_SHUTDOWN_NAME` verfügen. Dieses Privileg vergeben Sie über die lokalen Sicherheitseinstellungen (SYSTEMSTEUERUNG / VERWALTUNG / LOKALE SICHERHEITSRICHTLINIE / SICHERHEITSEINSTELLUNGEN / LOKALE RICHTLINIEN / ZUWEISEN VON BENUTZERRECHTEN), wo dieses Privileg als HERUNTERFAHREN DES SYSTEMS bezeichnet wird.

199 Die verfügbaren Leistungsindikatoren auslesen

Windows stellt ab Windows NT mit Leistungsindikatoren (Performance Counter) eine Möglichkeit zur Verfügung, die Leistung von Teilen des Systems zu überwachen. Mit vordefinierten Leistungsindikatoren können Sie z. B. den aktuellen Speicherverbrauch oder die CPU-Auslastung überprüfen. Unter Windows 2000 und XP können Sie die verfügbaren Leistungsindikatoren (u. a.) über den Systemmonitor (der in XP nur noch unter der Überschrift *Leistung* erscheint) überwachen. Diesen erreichen Sie über die Systemsteuerung (SYSTEMSTEUERUNG / VERWALTUNG / LEISTUNG bzw. SYSTEMSTEUERUNG / VERWALTUNG / SYSTEMMONITOR).

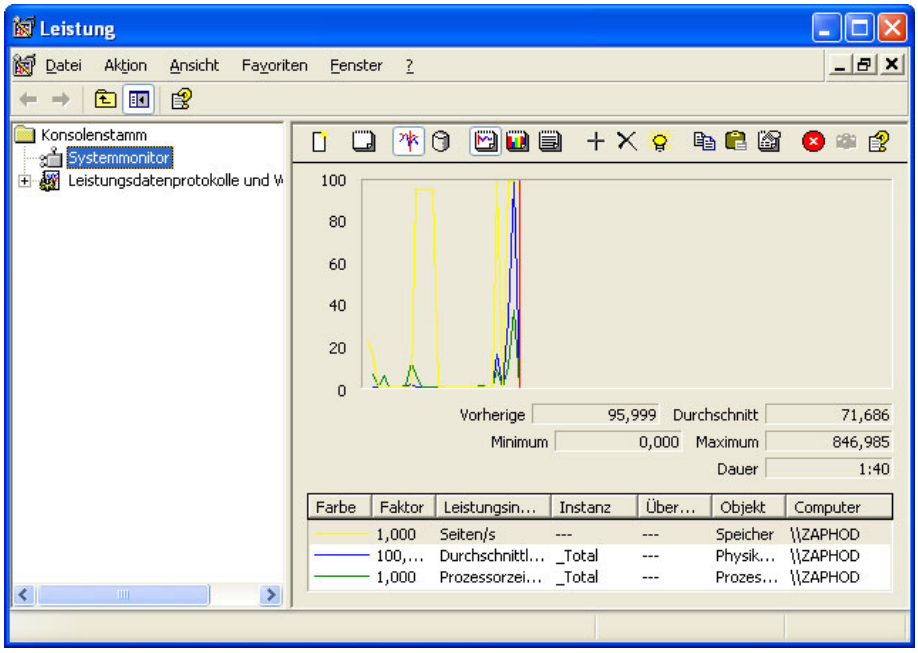


Abbildung 108: Der Systemmonitor von Windows XP

Über das Kontextmenü des Diagramms können Sie Leistungsindikatoren hinzufügen. Leistungsindikatoren sind in Kategorien eingeteilt. Die Kategorie *Speicher* enthält z. B. u. a. die Indikatoren *Verfügbare Bytes*, *Verfügbare KB*, *Verfügbare MB* und *Wechselfehler/s* (Abbildung 109).

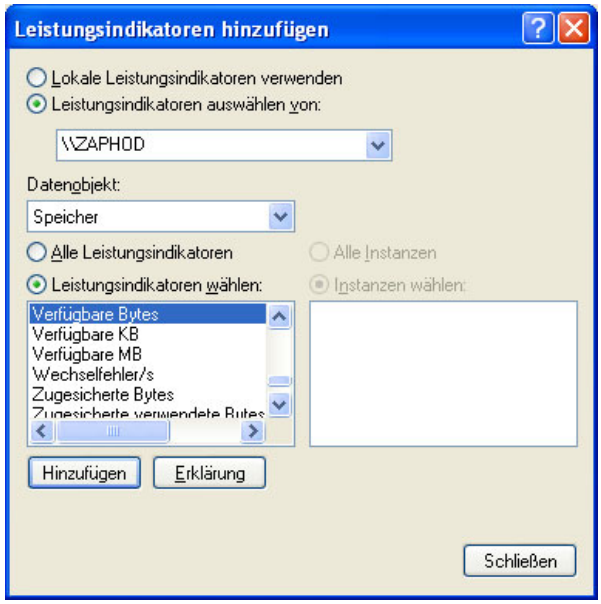


Abbildung 109: Hinzufügen einzelner Leistungsindikatoren

Der Systemmonitor öffnet die angegebenen Leistungsindikatoren und stellt diese so ein, dass sie in einem angegebenen Zeitintervall aktualisiert werden. Geöffnete Leistungsindikatoren werden lediglich im Arbeitsspeicher verwaltet und von Programmen wie dem Systemmonitor abgefragt. Diese Abfrage von Leistungsindikatoren können Sie auch in Ihre Programme integrieren, wie ich es ab dem Rezept 200 zeige.

Zunächst geht es aber darum, die verfügbaren Leistungsindikatoren aufzulisten. Zum Zugriff auf diese stellt das .NET Framework im Namensraum `System.Diagnostics` die Klassen `PerformanceCounterCategory` und `PerformanceCounter` zur Verfügung. Eine `PerformanceCounterCategory`-Instanz repräsentiert eine Leistungsindikatorenkategorie, eine `PerformanceCounter`-Instanz einen einzelnen Leistungsindikator. Über diese Klassen können Sie eine Liste der auf einem System verfügbaren Leistungsindikatoren erzeugen.

Hinweis

Beachten Sie, dass Leistungsindikatoren erst ab Windows NT zur Verfügung stehen. Unter Windows 98 und Me erhalten Sie bei der Verwendung der Klassen `PerformanceCounterCategory` und `PerformanceCounter` den Fehler »Dieses Feature erfordert Windows NT«.

Die Methode `WritePerformanceCounterNames` im folgenden Listing schreibt die verfügbaren Leistungsindikatoren in eine Textdatei, deren Dateiname übergeben wird. Sie erzeugt dazu zunächst eine `StreamWriter`-Instanz, ermittelt über die statische Methode `GetCategories` der `PerformanceCounterCategory`-Klasse alle verfügbaren Kategorien und geht diese durch. Nach-

dem der Name der Kategorie in den StreamWriter geschrieben wurde, liest WritePerformanceCounterNames über die Methode GetInstanceNames die Namen von eventuell vorhandenen »Instanzen« aus. Dazu sollten Sie wissen, dass Leistungsindikatoren ohne eine weitere Unterteilung in der Kategorie gespeichert sein oder in mehreren benannten Instanzen innerhalb der Kategorie vorkommen können. Verständlich wird dies durch den Vergleich zweier Kategorien. Die Kategorie *TCP* beinhaltet z.B. nur einen Satz an Leistungsindikatoren und somit keine Instanz. In der Kategorie *Prozess* werden aber einzelne Sätze der Leistungsindikatoren dieser Kategorie für jeden einzelnen Prozess und für die Gesamtheit aller Prozesse verwaltet. Diese Kategorie enthält also mehrere Instanzen der Leistungsindikatoren.

Dummerweise erzeugt die zum Auslesen der Leistungsindikatoren verwendete Methode GetCounters eines PerformanceCounterCategory-Objekts eine Ausnahme, wenn Sie diese Methode ohne Argument aufrufen und die Kategorie Instanzen beinhaltet. Für diesen Fall müssen Sie dieser Methode den Namen der Instanz übergeben und können dann die einzelnen Leistungsindikatoren dieser Instanz auslesen.

Zum Kompilieren der Methode müssen Sie die Namensräume System, System.Diagnostics, System.Text und System.IO importieren.

Da WritePerformanceCounterNames lediglich die Namen der Leistungsindikatoren einer Kategorie auflisten soll, wird überprüft, ob Instanzen bestehen. Ist dies der Fall, geht WritePerformanceCounterNames die Namen der Instanzen durch und schreibt diese in die Datei.

```
public static void WritePerformanceCounterNames(string filePath)
{
   // Datei zum Schreiben öffnen
   StreamWriter sr = new StreamWriter(filePath, false, Encoding.Default);

   // Ermitteln und Durchgehen der Leistungsindikatoren-Kategorien
   PerformanceCounterCategory[] pccList =
      PerformanceCounterCategory.GetCategories();
   foreach (PerformanceCounterCategory pcc in pccList)
   {
      sr.WriteLine("Kategorie: {0}", pcc.CategoryName);

      try
      {
         // Ermitteln der Namen der eventuell vorhandenen Instanzen
         string[] pcInstanceNames = pcc.GetInstanceNames();

         PerformanceCounter[] pcList = null;
         if (pcInstanceNames.Length > 0)
         {
            // Es existieren Instanzen der Kategorie: Zunächst die Namen der
            // Instanzen ausgeben
            sr.WriteLine("Instanzen von {0}:", pcc.CategoryName);
```

Listing 339: Abfragen der verfügbaren Leistungsindikatoren – Teil 1

```
            for (int i = 0; i < pcInstanceNames.Length; i++)
               sr.WriteLine("   " + pcInstanceNames[i]);
```

Listing 339: Abfragen der verfügbaren Leistungsindikatoren – Teil 1 (Forts.)

Für den Fall, dass Instanzen vorhanden sind, sollen die Indikatoren einer der Instanzen ausgelesen werden. Dabei entstand beim Testen aber das Problem, dass beim Aufruf der `GetCounters`-Methode für bestimmte Instanzen eine Ausnahme mit der Meldung erzeugt wurde, dass diese nicht vorhanden wäre (obwohl der Name ja von `GetInstanceNames` zurückgegeben wurde). Deshalb habe ich das Auslesen der Leistungsindikatoren in eine Schleife integriert, die alle Instanznamen durchgeht und für jede Instanz versucht, deren Indikatoren auszulesen. Für den Fall, dass alle Instanznamen zu Ausnahmen führen, speichert `WritePerformanceCounterNames` den zuletzt aufgetretenen Fehler in einer Variable, gibt diesen in die Datei aus und ruft `continue` auf, damit die äußere Schleife mit der nächsten Kategorie weitermacht.


```
            // Abfragen der Leistungsindikatoren einer der Instanzen
            bool ok = false;
            string lastError = null;
            foreach(string instanceName in pcInstanceNames)
            {
               try
               {
                  pcList  =
                     pcc.GetCounters(instanceName);
                  ok = true;
                  break;
               }
               catch (Exception ex)
               {
                  lastError = ex.Message;
               }
            }
            if (ok == false)
            {
               // Fehler ausgeben
               sr.WriteLine("Kann Kategorie {0} nicht abfragen, da " +
                  "beim Abfragen der Instanzen Fehler aufgetreten sind. " +
                  "Letzter Fehler: {1}", pcc.CategoryName, lastError);

               // Mit der nächsten Kategorie weitermachen
               sr.WriteLine();
               continue;
            }
         }
```

Listing 340: Abfragen der verfügbaren Leistungsindikatoren – Teil 2

Sind keine Instanzen vorhanden, ruft WritePerformanceCounterNames die GetCounters-Methode ohne Argument auf:

```
         else
         {
            pcList  = pcc.GetCounters();
         }
```

Listing 341: Abfragen der verfügbaren Leistungsindikatoren – Teil 3

Schließlich geht WritePerformanceCounterNames die ermittelten Leistungsindikatoren nur noch durch und schreibt deren Name, Beschreibung und den Typ, der ein Wert der Aufzählung PerformanceCounterType ist (siehe Tabelle 21), in die Datei:

```
         sr.WriteLine("Leistungsindikatoren von {0}:", pcc.CategoryName);
         foreach (PerformanceCounter pc in pcList)
         {
            sr.WriteLine("  {0}: {1} ({2})", pc.CounterName, pc.CounterHelp,
               pc.CounterType);
         }
      }
      catch (Exception ex)
      {
         sr.WriteLine("Kann Kategorie nicht abfragen: {0}", ex.Message);
      }

      sr.WriteLine();
   }

   // StreamWriter schließen
   sr.Close();
}
```

Listing 342: Abfragen der verfügbaren Leistungsindikatoren – Teil 4

Die äußere Ausnahmebehandlung habe ich übrigens deswegen integriert, da einige Leistungsindikatorenkategorien mit Instanzen scheinbar keine Daten speichern und bei der Abfrage zu der Ausnahme »Es können keine genauen Daten zurückgegeben werden, da die <Kategoriename>-Kategorie keine Instanzeninformationen enthält« führen.

Abbildung 110 zeigt das Ergebnis einer Abfrage der Leistungsindikatoren meines Systems in einem Editor.

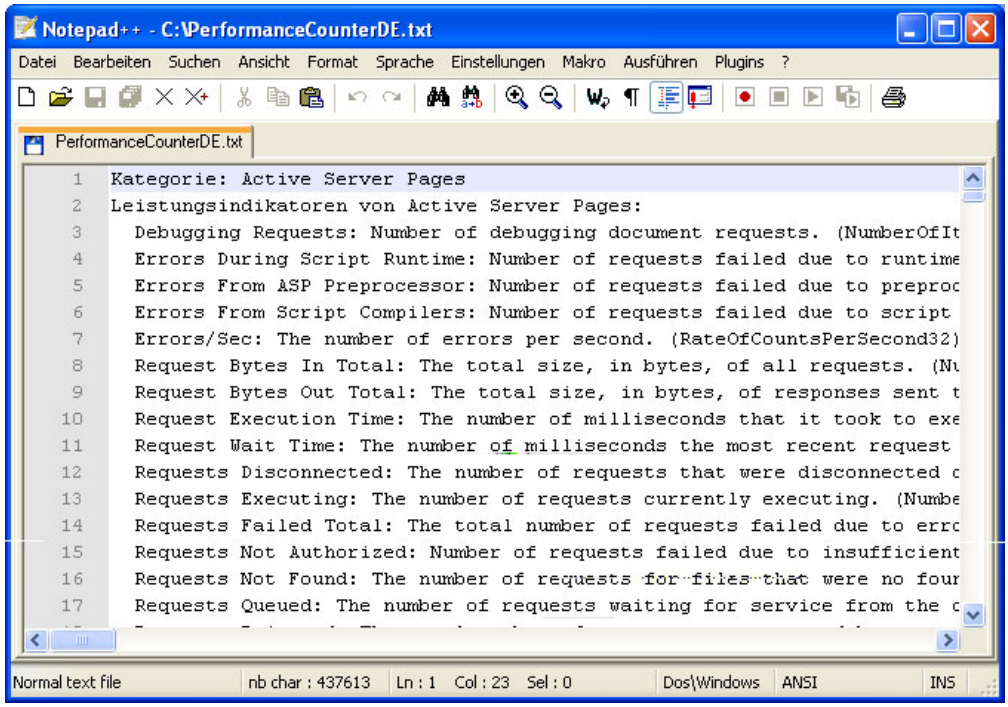


Abbildung 110: Das Ergebnis der WritePerformanceCounterNames-Methode in einem Texteditor

Der Typ eines Leistungsindikators sagt übrigens aus, welche Art von Informationen der Leistungsindikator speichert (siehe Tabelle 21).

Leistungsindikatortyp	Beschreibung
`AverageCount64`	Ein `AverageCount64`-Indikator gibt die durchschnittliche Anzahl von Aktionen oder Elementen innerhalb eines Messintervalls an. Ein Messintervall reicht vom letzten bis zum aktuellen Lesen des Wertes. Innerhalb dieses Intervalls kann der Indikatorwert auch mehrfach aktualisiert werden. Ein `AverageCount64`-Indikator berechnet den Durchschnitt aller Aktualisierungen im Messintervall. Ein Beispiel für einen solchen Indikator ist der Indikator *Mittlere Bytes/Übertragung* in der Kategorie *Datenträger*.
`AverageTimer32`	Dieser Typ beschreibt einen Leistungsindikator, der die für die Ausführung aller in einem Messintervall ausgeführten bestimmten Vorgänge benötigte durchschnittliche Zeit verwaltet. Der Indikator wird mit einer Zeitangabe in Form von Systemticks (Basisfrequenz des Motherboards) aktualisiert. Bei einer Messung (also beim Lesen des Indikatorwerts) liefert er den Durchschnitt der seit der letzten Messung geschriebenen Werte in Sekunden.
`ElapsedTime`	In einem Leistungsindikator dieses Typs wird eine seit einem Startzeitpunkt verstrichene Zeit in Sekunden gespeichert.

Tabelle 21: Die wichtigen Typen der Leistungsindikatoren

Leistungsindikatortyp	Beschreibung
`NumberOfItems32` `NumberOfItems64`	In einem solchen Leistungsindikator werden einfach nur Elemente oder Vorgänge gezählt. Ein Beispiel dafür ist der Indikator *Aufträge in Warteschlange* der Kategorie *Druckerwarteschlange*. `NumberOfItems32` definiert dabei einen 32-Bit-Integerwert, `NumberOfItems64` einen 64-Bit-Integerwert (was auch sonst ☺).
`RateOfCountsPerSecond32` `RateOfCountsPerSecond64`	Ein Leistungsindikator dieses Typs speichert die Anzahl von Vorgängen pro Sekunde als 32- oder 64-Bit-Integerzahl.
`Timer100Ns`	Ein Leistungsindikator dieses Typs speichert einen prozentualen Anteil der Aktivierungsdauer einer Komponente im Vergleich zu der Gesamtdauer eines Messintervalls. Die Zeit wird dabei in Einheiten von 100 Nanosekunden gemessen. Der Indikator *Prozessorzeit (%)* der *Prozess*-Kategorie ist ein Beispiel dafür.

Tabelle 21: Die wichtigen Typen der Leistungsindikatoren (Forts.)

Die Leistungsindikatoren, die einen Durchschnitt oder eine Differenz berechnen, sind vielleicht etwas schwer zu verstehen (bei meinen ersten Versuchen dachte ich jedenfalls, diese berechnen den gesamten Durchschnitt seit dem ersten Setzen eines Werts ☺). Diese Indikatoren berechnen den Durchschnitt bzw. die Differenz der Indikator-Rohwerte immer zwischen zwei Messungen (also zwischen dem letzten und dem aktuellen Lesen des Indikatorwerts). Wenn Sie einen `AverageCount64`-Indikator z. B. einmal pro Sekunde lesen, und dieser wird vom System innerhalb dieser Sekunde mit den Werten 1, 5 und 6 aktualisiert, ergibt das Lesen des Indikators den Wert 4 ((1 + 5 + 6) / 3). Nach dem Lesen wird der Indikator-Rohwert immer auf 0 gesetzt, weshalb beim nächsten Lesen wieder der Durchschnitt des dann aktuellen Messintervalls angezeigt wird.

200 Den Arbeitsspeicher überwachen

Die Überwachung des Arbeitsspeichers kann in einigen Situationen, bei denen eine (speicherintensive) Anwendung Probleme (z. B. Abstürze) verursacht, die nicht genauer lokalisiert werden können, ein wichtiges Mittel zur Problemlösung darstellen. Eine Speicherüberwachung können Sie natürlich über den Windows-Systemmonitor vornehmen. Viele Anwender (bei Ihren Kunden z. B.) sind damit aber überfordert. Deshalb ist es häufig sinnvoll, die Speicherüberwachung in das eigene Programm zu integrieren.

Die Standard-Windows-Leistungsindikatorenkategorie (siehe Rezept 199) *Speicher* (*Memory*) enthält einige Leistungsindikatoren zur Überwachung des Arbeitsspeichers. Die wichtigsten zeigt Tabelle 22.

Leistungsindikator	Beschreibung	Typ
Auslagerungsseiten: Residente Bytes (Pool Paged Resident Bytes)	die aktuelle Größe des Auslagerungsspeichers in Bytes	NumberOfItems64
Verfügbare Bytes (Available Bytes)	für Prozesse verfügbarer physikalischer Speicher in Bytes	NumberOfItems64

Tabelle 22: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie

Leistungsindikator	Beschreibung	Typ
Zugesicherte Bytes (Committed Bytes)	Zugesicherter virtueller Speicher in Bytes. Zugesicherter Speicher ist physikalischer Speicher, für den in Auslagerungsdateien Speicherplatz reserviert wurde.	NumberOfItems64
Zusagegrenze (Commit Limit)	virtueller Speicher in Bytes, der zugesichert werden kann, ohne die Auslagerungsdatei(en) zu vergrößern	NumberOfItems64

Tabelle 22: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie (Forts.)

Zur Überwachung des Arbeitsspeichers können Sie ein `PerformanceCounter`-Objekt verwenden, in dessen Konstruktor Sie den Namen der Leistungsindikatorenkategorie und den Namen des Leistungsindikators übergeben. Im optionalen dritten Argument können Sie mit `true` bestimmen, dass es auch erlaubt ist, in den Leistungsindikator zu schreiben, was allerdings nur bei benutzerdefinierten Leistungsindikatoren (siehe Rezept 203) Sinn macht.

Hinweis

Die Namen der Leistungsindikatorenkategorien und der Leistungsindikatoren können in der auf dem System aktuellen Sprache angegeben werden. Ein auf einem solchen System entwickeltes Programm läuft dann aber nicht mehr auf einem Computer mit einer anderen eingestellten Systemsprache. Laut meinen Versuchen und einem Newsgroup-Beitrag können Sie die Namen aber scheinbar immer auch in der englischen Sprache angeben. Damit sollte das Programm dann auf allen Systemen ausgeführt werden können. Um die englischen Texte zu erhalten, können Sie das Programm aus dem vorhergehenden Rezept vor dem Lesen der verfügbaren Indikatoren auf die englische Sprache umschalten:

```
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en");
```

Um den aktuellen Wert des Indikators abzufragen, rufen Sie die Methode `NextValue` auf, die einen `float`-Wert zurückgibt. Diesen Aufruf können Sie manuell (z. B. über einen Schalter) ausführen lassen oder in eine Timer-Methode oder einen Thread integrieren, um den Speicher regelmäßig und andauernd zu überwachen.

Listing 343 zeigt die Verwendung eines Threads zur Überwachung des Arbeitsspeichers in einer Konsolenanwendung. Das Programm startet in `Main` einen Thread mit der Methode `GetRAMUsage`. Diese Methode erzeugt zunächst ein `PerformanceCounter`-Objekt, liest in einer (Thread-typischen) Endlosschleife den aktuellen Wert des Leistungsindikators aus und gibt diesen an der Konsole aus. Beim Unterschreiten eines Minimalwerts (1 MB) wird zusätzlich eine Alarmmeldung ausgegeben. `GetRAMUsage` hält den Thread dann über `Thread.Sleep` für eine Sekunde an, um eine sekündliche Überprüfung des Arbeitsspeichers zu erreichen.

Die `Main`-Methode wartet, während der Thread läuft, so lange, bis der Anwender die Return-Taste betätigt, und beendet den Thread schließlich.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` einbinden. Das Programm kann nicht unter Windows 98 und Me ausgeführt werden, da Leistungsindikatoren auf diesen Systemen nicht zur Verfügung stehen.

```
/* Thread-Methode für die Überwachung des Arbeitsspeichers */
public static void GetRAMUsage()
{
   // PerformanceCounter-Instanz für den Leistungsindikator
   // 'Verfügbare Bytes' der Kategorie 'Speicher' erzeugen (in englischer
   // Sprache um das Programm auch auf Systemen ausführen zu können, die eine
   // andere Sprache als Deutsch verwenden)
   PerformanceCounter pc = new PerformanceCounter("Memory",
      "Available Bytes");
   do
   {
      // den nächsten Wert auslesen und ausgeben
      float byteCount = pc.NextValue();
      Console.WriteLine("{0:0} Bytes verfügbar", byteCount);

      // Alarmmeldung ausgeben, wenn der Minimalwert unterschritten wird
      if (byteCount < 1048576)
      {
         Console.WriteLine("ALARM: Der verfügbare Speicher " +
            "ist unter 1 MB gesunken");
      }

      // Thread eine Sekunde anhalten
      Thread.Sleep(1000);

   } while (true);
}

/* Main-Methode */
[STAThread]
static void Main(string[] args)
{
   Console.WriteLine("Beenden mit Return");

   // Thread starten
   Thread ramUsageThread = new Thread(new ThreadStart(GetRAMUsage));
   ramUsageThread.IsBackground = true;
   ramUsageThread.Start();

   // Auf Return warten
   Console.ReadLine();

   // Thread beenden
   ramUsageThread.Abort();
}
```

Listing 343: Überwachen der freien Bytes im physikalischen Arbeitsspeicher über einen Thread

Abbildung 111 zeigt das Programm in Aktion.

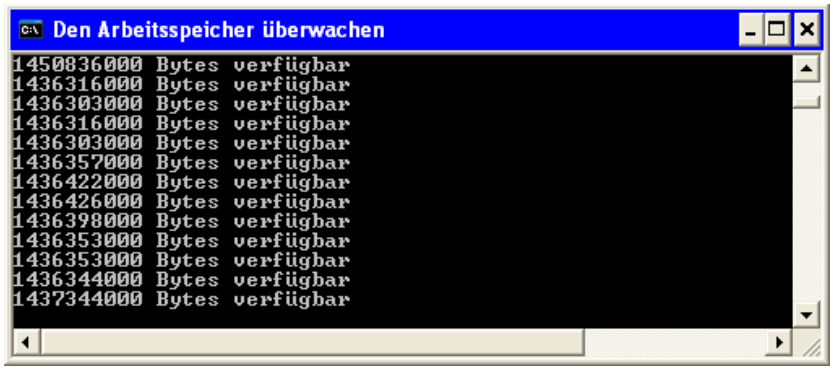


Abbildung 111: Das Programm zur Überwachung der freien Bytes im physikalischen Arbeitsspeicher in Aktion

201 Speicherauslastung und CPU-Belastung des aktuellen Prozesses überwachen

Bei speicher- oder prozessorintensiven Programmen ist es häufig zu Test- oder Debuggingzwecken sinnvoll, den vom Programm belegten Arbeitsspeicher oder die Prozessorauslastung in spezifischen Situationen (z. B. beim Aufruf einer rechenintensiven Methode) zu überwachen. Dazu können Sie die Leistungsindikatoren der Leistungsindikatorenkategorie *Prozess* verwenden, die ein standardmäßiger Bestandteil von Windows ist. Grundlegende Informationen zu Leistungsindikatoren finden Sie im Rezept 199. Tabelle 23 stellt die wichtigsten Leistungsindikatoren der Kategorie *Prozess* (*Process*) dar.

Leistungsindikator	Beschreibung	Typ
Arbeitsseiten (Working Set)	Die aktuelle Speicherbelegung für die Arbeitsseiten des Prozesses in Bytes. Aus Gründen der Performance wird Arbeitsspeicher immer auf so genannten Seiten verwaltet, die eine bestimmte Größe besitzen. Das Auslagern von zurzeit unbenutztem Speicher in die Auslagerungsdatei und das Wieder-Zurücklesen ist wesentlich effizienter, wenn an Stelle von einzelnen Bytes ganze Seiten geschrieben und gelesen werden können. Die Arbeitsseiten eines Prozessors beinhalten den kompletten vom Prozess belegten (virtuellen) Speicher (das ist der private und der mit anderen Prozessen geteilte Speicher). Die einem Prozess zugeordneten Arbeitsseiten können von Windows je nach dem freien Platz im virtuellen Speicher auch mehr Speicher beinhalten, als der Prozess benötigt.	NumberOfItems64
Auslagerungsdatei (Bytes) (Page File Bytes)	die aktuelle Anzahl der Bytes, die ein Prozess in der Auslagerungsdatei verwendet	NumberOfItems64

Tabelle 23: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie

Leistungsindikator	Beschreibung	Typ
Private Bytes (Private Bytes)	die aktuelle Größe des Speichers, der vom Prozess reserviert wurde und nicht mit anderen Prozessen geteilt werden kann	NumberOfItems64
Prozessorzeit (%) (% Processor Time)	der Prozentanteil der verstrichenen Zeit, in der alle Threads eines Prozesses den Prozessor verwendet haben	Timer100Ns
Threadanzahl (Thread Count)	die Anzahl der in diesem Prozess aktiven Threads	NumberOfItems32
Verstrichene Zeit (Elapsed Time)	die gesamte Zeitdauer, in Sekunden, in der dieser Prozess ausgeführt wurde	ElapsedTime
Virtuelle Größe (Virtual Bytes)	die aktuelle Größe des virtuellen Adressraums in Bytes, den der Prozess verwendet	NumberOfItems64

Tabelle 23: Die wichtigsten Leistungsindikatoren der Speicher-Kategorie (Forts.)

Die für uns aussagekräftigsten Indikatoren sind *Prozessorzeit (%)* und *Arbeitsseiten*. Dabei handelt es sich um die Werte, die auch im Windows-Taskmanager für jeden Prozess angezeigt werden. Der Indikator *Arbeitsseiten* steht für die Gesamt-Speicherausnutzung des Prozesses.

Das Programm in Listing 344 liest diese beiden Indikatoren und zusätzlich die Anzahl der aktiven Threads aus, um eine Aussage über die Systembelastung des aktuellen Prozesses machen zu können. Dazu werden zunächst drei `PerformanceCounter`-Instanzen für die Leistungsindikatoren erzeugt. Die Namen der Kategorie und der Indikatoren werden in englischer Sprache übergeben, um zu erreichen, dass das Programm auch auf Systemen ausgeführt werden kann, die eine andere Sprache als Deutsch verwenden.

Die Kategorie *Prozess* enthält immer mehrere Instanzen, nämlich eine Instanz pro ausgeführtem Prozess, die den Namen des Prozesses trägt, und eine Instanz mit dem Namen *_Total*, die Summen-Informationen für alle Prozesse liefert. Da das Programm die Daten des aktuellen Prozesses abfragen soll, liest es den Namen des aktuellen Prozesses aus der Eigenschaft `ProcessName` der `Process`-Instanz aus, die die Methode `Process.GetCurrentProcess` zurückgibt, und übergibt diesen Namen an die Eigenschaft `InstanceName` der erzeugten `PerformanceCounter`-Instanzen. Schließlich fragt das Programm den aktuellen Wert jeder dieser Instanzen ab und gibt diesen aus.

Zum Kompilieren des Programms müssen Sie die Namensräume `System` und `System.Diagnostics` importieren. Die Ausführung ist unter Windows 98 und Me nicht möglich, da Leistungsindikatoren erst ab Windows NT zur Verfügung stehen.

```
// PerformanceCounter-Instanzen für die zu überwachenden
// Leistungsindikatoren der Kategorie 'Prozess' erzeugen
PerformanceCounter pc1 = new PerformanceCounter("Process",
   "% Processor Time");
```

Listing 344: Anzeigen der vom aktuellen Prozess verwendeten Prozessorzeit, der Speicherauslastung und der Anzahl der Threads

```
PerformanceCounter pc2 = new PerformanceCounter("Process", "Working Set");
PerformanceCounter pc3 = new PerformanceCounter("Process", "Thread Count");

// Instanzname auf den Namen des aktuellen Prozesses festlegen
pc1.InstanceName = Process.GetCurrentProcess().ProcessName;
pc2.InstanceName = Process.GetCurrentProcess().ProcessName;
pc3.InstanceName = Process.GetCurrentProcess().ProcessName;

// Die jeweils nächsten Werte auslesen
float processorTime = pc1.NextValue();
float ramUsage = pc2.NextValue();
float threadCount = pc2.NextValue();

// Die Werte ausgeben
Console.WriteLine("Prozessorzeit: {0:0} %", processorTime);
Console.WriteLine("Speicherauslastung: {0:0} KB", ramUsage / 1024);
Console.WriteLine("Threadanzahl: {0:0}", threadCount);
```


Listing 344: Anzeigen der vom aktuellen Prozess verwendeten Prozessorzeit, der Speicherauslastung und der Anzahl der Threads (Forts.)

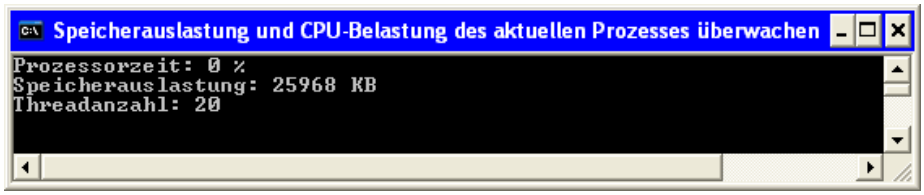


Abbildung 112: Das Beispielprogramm in Aktion

In einer realen Anwendung ist es wohl in der Regel sinnvoll, die Abfrage der Speicherauslastung vor, während und nach dem Aufruf von speicherintensiven Methoden auszuführen. Die Ermittlung der Prozessorauslastung würde hier aber keinen Sinn machen, da der aktuelle Thread beim Aufruf der `NextValue`-Methode nur diese ausführt und die damit ermittelte Prozessorauslastung keine Aussagekraft besitzt. Die Prozessorauslastung sollten Sie immer in einem separaten Thread ermitteln. Dort können Sie die ermittelten Daten z. B. in ein (Trace-) Protokoll (siehe Rezept 202) schreiben.

Vor und nach dem Aufruf prozessorintensiver Methoden können Sie diesen Aufruf ebenfalls im Protokoll dokumentieren und erhalten somit eine Zuordnung der Methodenaufrufe zu der CPU-Belastung. Im folgenden Rezept finden Sie eine Klasse, die diese Arbeit erleichtert.

202 Speicherauslastung und CPU-Belastung des aktuellen Prozesses in ein Trace-Protokoll schreiben

Das vorhergehende Rezept beschreibt, wie Sie die interessantesten Leistungsindikatoren der Leistungsindikatorenkategorie *Prozess* abfragen um Performance-Informationen für den Prozess einer Anwendung zu erhalten. Diese Informationen werden in der Praxis häufig in ein

Trace-Protokoll (»Spurverfolgungs-Protokoll«) geschrieben. Dieses Protokoll kann dann auf verschiedene Weisen ausgegeben (z. B. in eine Datei geschrieben) werden. Ein Beispiel dafür finden Sie in Listing 351, bei der Anwendung der im Folgenden beschriebenen Klasse.

Um das Tracen der Speicher- und der Prozessorauslastung zu erleichtern, habe ich die Klasse `PerformanceTracer` entwickelt. Über die Methode `TraceRamUsage` können Sie den aktuellen Speicherverbrauch der Anwendung in das Trace-Protokoll schreiben. Die Methode `StartCPUUsageTraceThread` startet einen Thread, der in dem in Millisekunden übergebenen Intervall die CPU-Belastung in das Trace-Protokoll schreibt. Die Methode `StopCPUUsageTraceThread` beendet diesen Thread.

Um die Klasse kompilieren zu können, müssen Sie die Namensräume `System`, `System.Threading` und `System.Diagnostics` einbinden. Das Programm kann nicht unter Windows 98 und Me ausgeführt werden, da Leistungsindikatoren auf diesen Systemen nicht zur Verfügung stehen.

Die Klasse enthält zunächst private Eigenschaften für die `ProcessCounter`-Instanzen, für die Übergabe des Abfrageintervalls an den CPU-Auslastungs-Thread und für den Thread selbst:


```
public class PerformanceTracer
{
   /* Private Felder für die PerformanceCounter */
   private PerformanceCounter ramCounter;
   private PerformanceCounter cpuCounter;

   /* Privates Feld zur Übergabe des Abfrageintervalls
      der CPU-Auslastung an den Thread */
   private int cpuUsageReadInterval;

   /* Privates Feld für den Thread */
   private Thread cpuUsageThread = null;
```

Listing 345: Beginn der Klasse zum Tracen der Speicher- und der CPU-Auslastung

Im Konstruktor der Klasse werden die `PerformanceCounter`-Instanzen für die Leistungsindikatoren *Working Set* und *% Processor Time* der *Process*-Kategorie erzeugt und mit dem Namen des aktuellen Prozesses initialisiert. Ich verwende hier die englischen Namen um zu erreichen, dass das Programm auch auf fremdsprachigen Systemen ausgeführt werden kann. Da unter Windows 2000 (auf meinen Testsystemen) nur die ersten 15 Zeichen des Prozessnamens in den Leistungsindikatoren verwaltet werden, fragt der Konstruktor ab, ob das Programm unter Windows 2000 ausgeführt wird und der Prozessname größer ist als 15 Zeichen. In diesem Fall wird der Name auf die erforderliche Länge gekürzt:

```
   public PerformanceTracer()
   {
      string processName = Process.GetCurrentProcess().ProcessName;
      if (Environment.OSVersion.Version.Major == 5 &&
```

Listing 346: Der Konstruktor der Klasse zum Tracen der Speicher- und der CPU-Auslastung

```
      Environment.OSVersion.Version.Minor == 0 &&
      processName.Length > 15)
   {
      // In Windows 2000 werden nur die ersten 15 Zeichen des
      // Prozessnamens verwaltet
      processName = processName.Substring(0, 15);
   }

   this.ramCounter = new PerformanceCounter("Process", "Working Set");
   this.ramCounter.InstanceName = processName;
   this.cpuCounter = new PerformanceCounter("Process",
      "% Processor Time");
   this.cpuCounter.InstanceName = processName;
}
```

Listing 346: Der Konstruktor der Klasse zum Tracen der Speicher- und der CPU-Auslastung (Forts.)

Die Methode `TraceRamUsage` ermittelt den aktuellen Speicherverbrauch über die `PerformanceCounter`-Instanz `ramCounter` und schreibt diesen mit der übergebenen Info in das Trace-Protokoll:

```
public void TraceRamUsage(string info)
{
   Trace.WriteLine(String.Format("Speicherauslastung bei {0}: " +
      "{1:0.00} KB ", info, this.ramCounter.NextValue() / 1024));
}
```

Listing 347: Methode zum Tracen der Speicherauslastung

Da die Überprüfung der aktuellen CPU-Auslastung in einem Thread erfolgen soll, enthält die Klasse die Methode `TraceCPUUsage`, die diese Arbeit erledigt. Das Auslesen der CPU-Auslastung erfolgt in einer Endlosschleife (die durch das Beenden des Threads abgebrochen werden kann). Über das `lock`-Schlüsselwort wird zunächst der Zugriff auf die `Trace`-Klasse für andere Threads gesperrt, damit keine Probleme bei einem gleichzeitigen Zugriff entstehen. Danach liest `TraceCPUUsage` über das private `PerformanceCounter`-Objekt `cpuCounter` die aktuelle Prozessorauslastung aus und schreibt diese in das Trace-Protokoll. Da das Lesen in Intervallen erfolgen soll, hält die Methode den Thread dann noch für das angegebene Intervall an:

```
private void TraceCPUUsage()
{
   do
   {
      lock(typeof(Trace))
      {
         Trace.WriteLine("CPU-Auslastung " +
```

Listing 348: Methode für den Thread zum Tracen der CPU-Auslastung

```
                DateTime.Now.ToLongTimeString() + ": " +
                this.cpuCounter.NextValue() + "%");
            }

            Thread.Sleep(this.cpuUsageReadInterval);

        } while (true);
    }
```

Listing 348: Methode für den Thread zum Tracen der CPU-Auslastung (Forts.)

Die Methode `StartCPUUsageTraceThread` schreibt zunächst das übergebene Intervall in das private Feld `cpuUsageReadInterval`, erzeugt dann einen neuen Thread für die `TraceCPUUsage`-Methode und startet diesen:

```
    public void StartCPUUsageTraceThread(int readInterval)
    {
        this.cpuUsageReadInterval = readInterval;

        this.cpuUsageThread = new Thread(new ThreadStart(this.TraceCPUUsage));
        cpuUsageThread.Start();
    }
```

Listing 349: Methode zum Starten des Thread, der die CPU-Auslastung verfolgt

Die Methode `StopCPUUsageTraceThread` ruft die `Abort`-Methode des Thread auf, um diesen zu stoppen:

```
    public void StopCPUUsageTraceThread()
    {
        this.cpuUsageThread.Abort();
    }
}
```

Listing 350: Methoden zum Stoppen des Thread, der die CPU-Auslastung verfolgt

Listing 351 zeigt eine beispielhafte Anwendung dieser Klasse in einer Konsolenanwendung. Die Methode `Stress` erzeugt in einer Schleife mit 100.000 Durchläufen einzelne `DateTime`-Objekte und gibt den Schleifenzähler an der Konsole aus. Die Schleife produziert eine hohe CPU-Auslastung, das Instanzieren der `DateTime`-Objekte führt zu einer erhöhten Speicherauslastung, da der Garbage Collector diese frühestens nach der Ausführung der Schleife freigibt. Beim Ein- und Austritt schreibt `Stress` den aktuellen Speicherverbrauch über die `TraceRamUsage`-Methode in das Trace-Protokoll.

Die `Main`-Methode initialisiert zunächst das Trace-Protokoll mit einem standardmäßigen Trace-Listener, der die erhaltenen Daten in eine Textdatei schreibt. Dann erzeugt `Main` eine Instanz der `PerformanceTracer`-Klasse (die in einer privaten statischen Variablen verwaltet

wird). Über diese Instanz startet `Main` dann den Thread zur Messung der CPU-Auslastung und ruft die `Stress`-Methode auf. Nach der Rückkehr dieser Methode beendet `Main` den Thread und schließt das Trace-Protokoll, um die Protokolldatei zu schreiben.

```
using System;
using System.IO;
using System.Diagnostics;
using Addison_Wesley.Codebook.System;

namespace Prozess_Performance_Tracing
{
   class Start
   {
      // Feld für den Performance-Tracer
      private static PerformanceTracer performanceTracer;

      /* Methode, die eine hohe CPU-Auslastung erzeugt */
      public static void Stress()
      {
         // Protokollieren des Eintritts in die Methode
         performanceTracer.TraceRamUsage("Eintritt in Stress");

         // Prozessor- und speicherintensive Schleife
         for (int i = 0; i < 100000; i++)
         {
            // DateTime-Objekt zur Simulation von Speicherauslastung erzeugen
            DateTime d = new DateTime(0);

            string dateString = d.ToString();
            Console.WriteLine(dateString);
         }

         // Protokollieren des Austritts aus der Methode
         performanceTracer.TraceRamUsage("Austritt in Stress");
      }

      [STAThread]
      static void Main(string[] args)
      {
         // Protokolldatei löschen, falls diese vorhanden ist
         string logfileName = "c:\\trace.log";
         if (File.Exists(logfileName))
            File.Delete(logfileName);
```

Listing 351: Beispielhafte Anwendung der Klasse zum Tracen der Speicher- und der CPU-Auslastung in einer Konsolenanwendung

```
            // Trace-Listener zum Schreiben in eine Datei erzeugen und
            // an die Listeners-Auflistung der Trace-Klasse anfügen
            Trace.Listeners.Clear();
            Trace.Listeners.Add(new System.Diagnostics.TextWriterTraceListener(
               logfileName));

            // PerformanceTracer-Instanz erzeugen
            performanceTracer = new PerformanceTracer();

            // CPU-Auslastungs-Thread mit einem 100ms-Intervall starten
            performanceTracer.StartCPUUsageTraceThread(100);

            // Speicher- und prozessorintensive Methode aufrufen
            Stress();

            // CPU-Auslastungs-Thread stoppen
            performanceTracer.StopCPUUsageTraceThread();

            // Trace-Listener schließen
            Trace.Listeners[0].Close();
        }
    }
}
```

Listing 351: Beispielhafte Anwendung der Klasse zum Tracen der Speicher- und der CPU-Auslastung in einer Konsolenanwendung (Forts.)

Abbildung 113 zeigt eine durch das Beispielprogramm erzeugte Protokolldatei in einem Editor. Die Datei habe ich ein wenig manipuliert, sodass der Anfang und das Ende des Protokolls sichtbar sind.

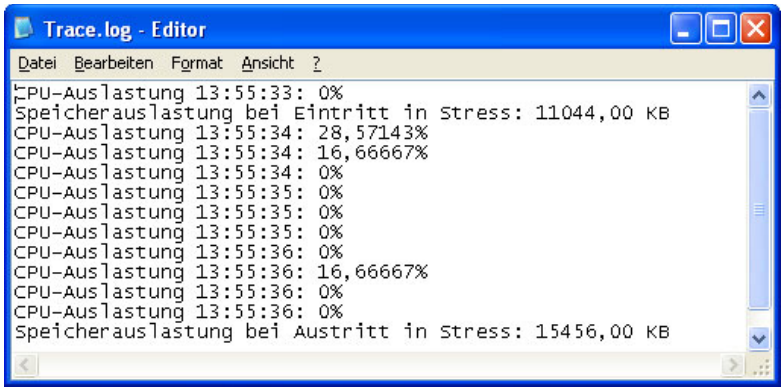


Abbildung 113: Die durch das Beispielprogramm erzeugte Trace-Protokolldatei

203 Eigene Leistungsindikatoren implementieren

In manchen Fällen ist es sinnvoll, bestimmte Leistungsdaten von Programmen von außen zu überwachen. Wenn Sie z. B. in einer Datenbankanwendung mit Transaktionen arbeiten, könnten Sie die Gesamtanzahl der Transaktionen und die Anzahl der aktuell offenen Transaktionen überwachen. Wenn Sie dazu eigene Leistungsindikatoren verwenden, können Sie diese Daten z. B. über den Systemmonitor überwachen (siehe Rezept 199).

Eigene Leistungsindikatoren können Sie über den Server-Explorer von Visual Studio 2008 erzeugen. Wählen Sie dazu im Kontextmenü des Leistungsindikatoren-Eintrags (SERVER / <RECHNERNAME> / LEISTUNGSINDIKATOREN) den Befehl NEUE KATEGORIE ERSTELLEN. In der Praxis ist es aber sinnvoller, eigene Leistungsindikatoren im Programm zu erstellen, damit dieses problemlos auch auf Rechnern ausgeführt werden kann, die diese Leistungsindikatoren noch nicht besitzen.

Zur Erzeugung eigener Leistungsindikatoren müssen Sie zunächst eine eigene Leistungsindikatorenkategorie erstellen. Dazu ist wichtig zu wissen, dass Sie in bereits vorhandene Kategorien keine Leistungsindikatoren hinzufügen können. Das gilt sogar dann, wenn Sie die Kategorie selbst erzeugt haben. Sie können aber natürlich überprüfen, ob die Leistungsindikatorenkategorie bereits existiert, oder diese vor dem Erzeugen einfach löschen. Eigene Leistungsindikatoren und deren Kategorien bleiben innerhalb von Windows so lange bestehen, bis diese explizit gelöscht werden.

Zur Erzeugung einer Leistungsindikatorenkategorie rufen Sie die statische `Create`-Methode der `PerformanceCounterCategory`-Klasse aus dem Namensraum `System.Diagnostics` auf. Am ersten Argument übergeben Sie einen systemweit eindeutigen Namen für die Kategorie. Wenn Sie den Namen einer bereits existierenden Kategorie verwenden, resultiert `Create` in einer Ausnahme vom Typ `InvalidOperationException`. Verwenden Sie vielleicht einfach den Namen Ihrer Anwendung als Kategoriename.

Am zweiten Argument können Sie eine Beschreibung für diese Kategorie übergeben. Einer Überladung der `Create`-Methode können Sie dann noch den Namen eines Leistungsindikators und dessen Beschreibung übergeben. Da Sie damit aber wenig flexibel sind, sollten Sie die Überladung verwenden, die am vierten Argument eine Instanz der Klasse `CounterCreationDataCollection` erwartet. Diese Auflistung verwaltet Instanzen der Klasse `CounterCreationData`, die die zur Erstellung eines Leistungsindikators notwendigen Informationen speichern. Damit können Sie dann auch beliebig viele Leistungsindikatoren in der Kategorie erstellen. Der Konstruktor der `CounterCreationData`-Klasse erwartet am ersten Argument den Namen des Leistungsindikators, am zweiten eine Beschreibung (die z. B. im Systemmonitor angezeigt werden kann) und am dritten den Typ des Indikators (als Wert der Aufzählung `PerformanceCounterType`). Die wichtigen Typen habe ich bereits im Rezept 199 beschrieben.

Am dritten Argument der `Create`-Methode der `PerformanceCounterCategory`-Klasse übergeben Sie dann noch eine Information darüber, ob der Performance-Counter in einer oder in mehreren Instanzen vorkommen kann. Dazu übergeben Sie die Werte der `PerformanceCounterCategoryType`-Aufzählung.

Der Typ des Leistungsindikators bestimmt, wie dessen Wert beim Abruf über `NextValue` berechnet wird. Den rohen Wert eigener Leistungsindikatoren können Sie später zu beliebigen Zeitpunkten auf einen bestimmten Wert setzen oder einfach nur um einen bestimmten Wert erhöhen. Beim Abruf des berechneten Werts wird der Wert eines Leistungsindikators vom Typ `NumberOfItems32` z. B. einfach zurückgegeben. Andere Leistungsindikatoren wie z. B. `AverageTimer32` berechnen ihren Wert etwas komplizierter. Die Art der Berechnung wird bei der Dokumentation der `PerformanceCounterType`-Aufzählung recht gut beschrieben (wenn diese auch teilweise komplex ist). Ein `AverageTimer32`-Indikator berechnet z. B. die durchschnittliche Dauer einer Operation, die innerhalb eines Messintervalls u. U. mehrfach aufgerufen wird. Für einen solchen Indikator benötigen Sie einen Basis-Indikator vom Typ `AverageBase` und den `AverageTimer32`-Indikator selbst. Der `AverageBase`-Indikator, der dem `AverageTimer32`-Indikator direkt folgen muss, verwaltet die Gesamtanzahl der Operationen und muss bei jeder neuen Operation um 1 erhöht werden. Den `AverageTimer32`-Indikator aktualisieren Sie dann mit der gemessenen Zeitdauer der Operation in Systemticks (ein Systemtick ist das Basisintervall des Motherboard-Zeitgebers). Der Wert eines solchen Indikators wird automatisch in Sekunden umgerechnet.

Achtung

Ein `AverageTimer32`-Indikator ist ein gutes Beispiel für eine schlechte Dokumentation. Bei diesem Indikator (und auch bei anderen, die Systemticks einsetzen) müssen Sie darauf achten, dass der Indikator mit Systemticks arbeitet und nicht, wie in vielen falschen Beispielen für diesen Indikator zu sehen, mit den Ticks des `DateTime`-Datentyps (100 Nanosekunden). Wenn Sie mit einem `AverageTimer32`-Indikator arbeiten, müssen Sie die Systemticks über die API-Funktion `QueryPerformanceCounter` ermitteln.

Listing 352 zeigt eine beispielhafte Verwendung eigener Leistungsindikatoren. Das Programm überprüft zunächst, ob die zu erzeugende Leistungsindikatorenkategorie (die den Namen der Anwendung trägt) bereits existiert, und löscht die Kategorie in diesem Fall. Damit wird sichergestellt, dass auch nachträgliche Änderungen an den Leistungsindikatoren problemlos im Quellcode vorgenommen werden können. Falls die Indikatoren nicht mehr geändert werden, können Sie die Kategorie auch einfach bestehen lassen und nur dann erzeugen, wenn sie noch nicht existiert.

Dann erstellt das Programm die Kategorie. Dazu wird zunächst eine Instanz der Klasse `CounterCreationDataCollection` erzeugt und mit den Daten für vier neue Leistungsindikatoren gefüllt. Die ersten zwei Indikatoren sind einfache Zähl-Indikatoren. Der dritte soll allerdings die mittlere Anzahl von offenen Transaktionen verwalten und ist deshalb vom Typ `AverageCount64`. Da dieser Indikator einen Basis-Indikator vom Typ `AverageBase` benötigt, der später mit der Gesamtanzahl der Zählvorgänge aktualisiert werden muss, wird dieser ebenfalls erzeugt und der Auflistung hinzugefügt. Über die `Create`-Methode der `PerformanceCounterCategory`-Klasse erzeugt das Programm dann die Kategorie.

Zum Kompilieren müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Diagnostics` und `System.Windows.Forms` importieren. Zur Ausführung des Programms ist Windows NT, Windows 2000, XP oder eine neuere Windows-Version notwendig.

```
// Ermitteln, ob die Leistungsindikatoren-Kategorie bereits existiert,
// und Löschen derselben, falls dies der Fall ist
if (PerformanceCounterCategory.Exists(Application.ProductName))
{
   PerformanceCounterCategory.Delete(Application.ProductName);
}

// CounterCreationDataCollection-Instanz für die Indikatorendaten der zu
// erzeugenden Leistungsindikatoren-Kategorie erzeugen und mit den Daten von
// drei Leistungsindikatoren (plus einem Basisindikator) füllen
CounterCreationDataCollection ccdCol = new CounterCreationDataCollection();
ccdCol.Add(new CounterCreationData("Gesamtanzahl der Transaktionen",
   "Verwaltet die Anzahl der insgesamt ausgeführten Transaktionen",
   PerformanceCounterType.NumberOfItems32));
ccdCol.Add(new CounterCreationData("Offene Transaktionen",
   "Zeigt die Anzahl der aktuell offenen Transaktionen an",
   PerformanceCounterType.NumberOfItems32));
ccdCol.Add(new CounterCreationData("Offene Transaktionen, Durchschnitt",
   "Zeigt die durchschnittlichen offenen Transaktionen",
   PerformanceCounterType.AverageCount64));
ccdCol.Add(new CounterCreationData(
   "Offene Transaktionen, Durchschnitt, Basis",
   "Basis für den Counter 'Offene Transaktionen, Durchschnitt'",
   PerformanceCounterType.AverageBase));

// Leistungsindikatoren-Kategorie erzeugen
PerformanceCounterCategory.Create(Application.ProductName,
   "Demo für Leistungsindikatoren",
   PerformanceCounterCategoryType.SingleInstance, ccdCol);
```

Listing 352: Erzeugen einer eigenen Leistungsindikatorkategorie mit Leistungsindikatoren

Nun können Sie die Leistungsindikatoren referenzieren und aktualisieren. Dazu erzeugen Sie Instanzen der Klasse `PerformanceCounter`, denen Sie im Konstruktor den Kategorie- und den Leistungsindikatornamen übergeben. Am dritten Argument übergeben Sie `false`, damit der Indikator nicht schreibgeschützt ist.

```
PerformanceCounter pc1 = new PerformanceCounter(Application.ProductName,
   "Gesamtanzahl der Transaktionen", false);
PerformanceCounter pc2 = new PerformanceCounter(Application.ProductName,
   "Offene Transaktionen", false);
PerformanceCounter pc3 = new PerformanceCounter(Application.ProductName,
   "Offene Transaktionen, Durchschnitt", false);
PerformanceCounter pc4 = new PerformanceCounter(Application.ProductName,
   "Offene Transaktionen, Durchschnitt, Basis", false);
```

Listing 353: Referenzieren der eigenen Leistungsindikatoren

Den Wert von Leistungsindikatoren können Sie über die `Increment`- oder `IncrementBy`-Methode erhöhen bzw. über `IncrementBy` oder `Decrement` erniedrigen. Den rohen Wert des Indikators können Sie über die Eigenschaft `RawValue` lesen und setzen. Das folgende Beispiel zeigt dies für die Demo-Indikatoren. Das Programm erhöht in einer Schleife jeweils den Indikator für die Gesamtanzahl der Transaktionen um den Wert 1 und erzeugt für die Anzahl der offenen Transaktionen einen Zufallswert. Der Indikator für die Ermittlung des Durchschnitts der offenen Transaktionen wird über `IncrementBy` um die aktuelle Anzahl der offenen Transaktionen erhöht. Da dieser Indikator den Typ `AverageCount64` besitzt, wird der Basis-Indikator (`pc4`) dann noch um den Wert 1 erhöht. Damit das Ganze im Systemmonitor beobachtbar ist, ruft das Programm am Ende der Schleife `Thread.Sleep` auf, um den aktuellen Thread für eine halbe Sekunde anzuhalten. Das Beispiel setzt übrigens den Indikator für die Gesamtanzahl der Transaktionen vor der Schleife zurück, da in meinen Versuchen der alte Wert des Indikators auch dann bestehen blieb, nachdem die Kategorie gelöscht und neu erzeugt wurde.


```
// Gesamtanzahl der Transaktionen zurücksetzen
pc1.RawValue = 0;

// Aktualisieren der eigenen Leistungsindikatoren
Random random = new Random();
for (int i = 0; i < 100; i++)
{
   Console.WriteLine("Inkrementierung " + i + " ...");
   int openTransactionCount = random.Next(10);
   // Gesamtanzahl der Transaktionen erhöhen
   pc1.Increment();
   // Anzahl der offenen Transaktionen angeben
   pc2.RawValue = openTransactionCount;
   // Durchschnitt der offenen Transaktionen erhöhen
   pc3.IncrementBy(openTransactionCount);
   // Basis-Zähler für den Indikator pc3 erhöhen
   pc4.Increment();

   System.Threading.Thread.Sleep(500);
}
```

Listing 354: Aktualisieren von Leistungsindikatoren

Im Windows-Systemmonitor (siehe Rezept 199) können Sie nun Ihre Leistungsindikatoren hinzufügen (Abbildung 114).

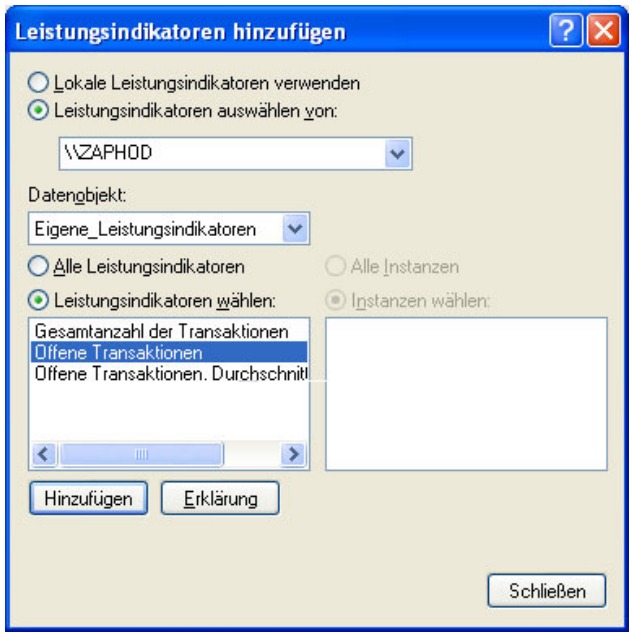


Abbildung 114: Hinzufügen der eigenen Leistungsindikatoren im Systemmonitor. Das Programm, das die Indikatoren erzeugt hat, besitzt den Namen »Eigene Leistungsindikatoren«

Wenn Sie das Programm dann ausführen, können Sie die Leistungsindikatoren im Systemmonitor überwachen (Abbildung 115).

Beachten Sie, dass der Indikator *Offene Transaktionen, Durchschnitt* vom Typ `AverageCount64` ist. Ein solcher Indikator berechnet (wie andere Durchschnitts-Indikatoren) bei einer Messung (also beim *Lesen* des Indikatorwerts, nicht beim *Schreiben* des rohen Werts!) den Durchschnitt aller Werte, die seit der vorhergehenden Messung geschrieben wurden. Er berechnet also den Durchschnitt der Aktionen im gegebenen Messintervall.

Erfolgt die Messung z. B. jede Sekunde und werden innerhalb dieser Sekunden die drei Werte 1, 5 und 6 geschrieben, ergibt das Lesen dieses Indikators den Wert 4. Daraus lässt sich erkennen, warum ein solcher Indikator über `IncrementBy` aktualisiert werden muss und warum ein zugeordneter `AverageBase`-Indikator die Anzahl der Werte verwaltet (wobei die folgenden Ausführungen lediglich eine Vermutung darstellen): Nach einer Messung setzt ein solcher Indikator den Wert auf 0 und speichert den aktuellen Wert des Basisindikators zwischen. `IncrementBy` erhöht den Wert dann ja wieder, u. U. bis zu nächsten Messung mehrfach. Bei einer Messung berechnet der Indikator einfach die Differenz zwischen dem aktuellen und dem letzten Wert des Basisindikators und teilt seinen Wert durch die so ermittelte Anzahl. Das ist doch einfach, oder (da frage ich mich, warum ich stundenlang in der Dokumentation und im Internet nach der Funktionsweise solcher Indikatoren gesucht habe ☺).

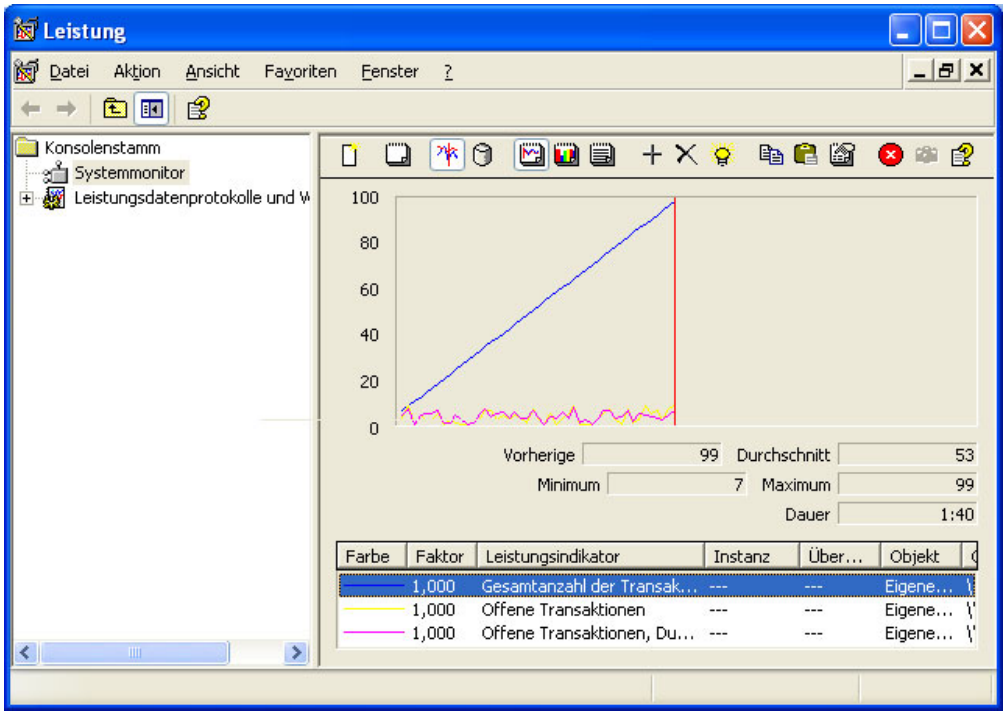


Abbildung 115: Die eigenen Leistungsindikatoren im Systemmonitor

204 Monitor abschalten, in den Energiesparmodus schalten und wieder einschalten

Einen Monitor, der Energiespar-Features unterstützt, können Sie über die Windows-API-Funktion `SendMessage` abschalten, in den Energiesparmodus umschalten und wieder einschalten. Der Monitor und der Treiber der Grafikkarte müssen dazu allerdings das Ab- und Umschalten unterstützen.

`SendMessage` sendet Windows-Nachrichten an einzelne oder an alle Top-Level-Fenster. Am ersten Argument übergeben Sie entweder den Handle des Zielfensters oder die Konstante `HWND_BROADCAST` (0xFFFF) zum Senden an alle Top-Level-Fenster.

Die zu sendenden Nachrichten werden am zweiten Argument über im Windows-API definierte `int`-Konstanten festgelegt. Die Konstante `WM_SYSCOMMAND` (0x0112) steht hier für das Senden einer Systemnachricht (die vom Anwender normalerweise über die Betätigung des Systemmenüs eines Fensters gesendet wird).

Über das dritte und vierte Argument übergeben Sie `SendMessage` Parameter für die Nachricht, zumeist wieder über im Windows-API definierte Konstanten. Um den Monitor ab-, um- oder einzuschalten, übergeben Sie am dritten Argument die Konstante `SC_MONITORPOWER` (0xF170). Bei dieser speziellen Nachricht bestimmt das vierte Argument, ob der Monitor abgeschaltet (2), in den Energiesparmodus umgeschaltet (1) oder eingeschaltet (-1) wird.

Da SendMessage Nachrichten immer an Fenster sendet, können Sie die Nachricht entweder an das Hauptfenster Ihrer Anwendung senden, indem Sie den Handle dieses Fensters aus der Eigenschaft Handle auslesen und an SendMessage übergeben. Um auf diese eigentlich unnötige Übergabe eines Fenster-Handles zu verzichten, können Sie die Nachricht stattdessen einfach an alle Top-Level-Fenster senden, indem Sie am ersten Argument HWND_BROADCAST übergeben. Da es sich dabei um eine int-Konstante handelt und SendMessage einen IntPtr-Typ erwartet, müssen Sie die Konstante in einen solchen konvertieren.

Die Klasse Monitor in Listing 355 implementiert das Abschalten, Umschalten und das Ausschalten des Monitors in statischen Methoden. Um diese Klasse kompilieren zu können, müssen Sie die Namensräume System und System.Runtime.InteropServices importieren.


```
public class Monitor
{
   // Deklaration der API-Funktion SendMessage
   [DllImport("User32.dll")]
   private static extern int SendMessage(IntPtr handle, int msg, int
      wparam, int lparam);

   // Deklaration der benötigten Konstanten
   private const int WM_SYSCOMMAND = 0x0112;
   private const int SC_MONITORPOWER = 0xF170;
   private const int HWND_BROADCAST = 0xFFFF;

   /* Schaltet den Monitor ab */
   public static void TurnOff()
   {
      SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 2);
   }

   /* Schaltet den Monitor in den Energiesparmodus um */
   public static void SwitchToLowPower()
   {
      SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 1);
   }

   /* Schaltet einen abgeschalteten Monitor wieder ein */
   public static void TurnOn()
   {
      SendMessage((IntPtr)HWND_BROADCAST, WM_SYSCOMMAND,
         SC_MONITORPOWER, -1);
   }
}
```

Listing 355: Methoden zum Ein-, Um- und Ausschalten des Monitors

Hinweis

Beim Test unter Windows 98 ließ sich der Monitor zwar abschalten, nicht aber (über einen Timer) wieder einschalten. Nach dem manuellen Einschalten des Monitors war das Programm abgestürzt und musste über den Taskmanager beendet werden. Die Fehlerursache habe ich nicht ausfindig machen können.

205 System-Hotkeys registrieren und auswerten

Über die API-Funktion `RegisterHotkey` können Sie einen Hotkey (eine Tastenkombination, die zu einer Aktion führt) mit einem Fenster verbinden. Dazu können Sie im Prinzip alle möglichen Tastenkombination verwenden (außer vom System reservierte wie z. B. F12. `RegisterHotkey` wird dazu am dritten Argument eine Angabe der Modifizier-Tasten ALT, STRG, SHIFT, Windows-Taste) und am vierten Argument die eigentliche Taste übergeben. Die Modifizier-Tasten können miteinander (über |) kombiniert werden um komplexe Tastenkombinationen abzubilden.

Wird die Tastenkombination betätigt, ruft Windows die Fenster-Funktion des mit dem Hotkey verbundenen Fensters auf und übergibt die Nachricht `WM_HOTKEY` (0x312). Das Fenster muss diese Nachricht auswerten und entsprechend reagieren, z. B., indem das Fenster sich in den Vordergrund setzt.

System-Hotkeys sind immer aktiv, unabhängig von der gerade im Vordergrund stehenden Anwendung. Auf diese Weise ist z. B die Tastenkombination ALT + TAB mit dem Windows-Task-Wechsler verknüpft.

Achtung

Beachten Sie, dass Tastenkombinationen in Anwendungen nicht mehr funktionieren, wenn Sie diese Tastenkombination als System-Hotkey registrieren. Wenn Sie z. B. in einer Anwendung F5 registrieren und die Anwendung starten, funktioniert die F5-Taste u. a. in Visual Studio nicht mehr, sondern startet die in Ihrer Anwendung programmierte Aktion für den Hotkey.

Wird die Anwendung, die einen Hotkey registriert hat, geschlossen, muss diese den Hotkey mit `UnregisterHotkey` wieder freigeben. Jede Anwendung kann beliebig viele Hotkeys registrieren, allerdings sollte (laut der Dokumentation) jedem Hotkey eine für den registrierenden Task eindeutige Integer-ID übergeben werden. Diese ID wird der `WM_HOTKEY`-Nachricht in dem Feld `WParam` der übergebenen `Message`-Instanz übergeben und kann dort ausgewertet werden.

Hinweis

Die Dokumentation von `RegisterHotkey` gibt an, dass die ID für den aufrufenden Thread eindeutig sein *sollte* (warum, wird nicht genannt). Wahrscheinlich wird das so beschrieben, da die ID der `WM_HOTKEY`-Nachricht übergeben wird und in `WndProc` von der Anwendung ausgewertet werden kann. Meine Versuche mit mehreren gleichzeitig reservierten Hotkeys mit derselben ID (bei denen ich die ID nicht ausgewertet habe) haben bisher keine Probleme gezeigt. Da die ID aber auch `UnregisterHotkey` übergeben wird, ist es u. U. wichtig, eine wirklich eindeutige ID zu verwenden.

Zudem sollte die ID für Anwendungen im Bereich von 0 bis 0xBFFF (49151) liegen, für klassische DLLS (die wir ja gar nicht entwickeln können) im Bereich von 0xC000 (49152) bis 0xFFFF (65535). Versuche mit wesentlich größeren IDs haben bisher aber auch keine Probleme gezeigt. Trotzdem halte ich mich an die Dokumentation und verwende eine ID im Bereich zwischen 0 und 49151.

War der Hotkey zuvor bereits mit einer anderen Anwendung verknüpft, wird er temporär überschrieben. Gibt die Anwendung den Hotkey wieder frei, gilt wieder die vorherige Verknüpfung. So können Sie zum Beispiel beim Start einer Anwendung die Tastenkombination ALT + TAB als Hotkey registrieren und damit den Windows-Task-Wechsler so lange ersetzen, wie die Anwendung ausgeführt wird (das war der Grund, warum ich dieses Rezept entwickelt habe ☺).

Hinweis

Beachten Sie, dass die F12-Taste (ohne Modifizierer) nicht als Hotkey-Taste verwendet werden kann. Windows hat diese Taste für den System-Debugger reserviert. Wenn Sie diese Taste als Hotkey-Taste verwenden, resultiert ein Fehler und `RegisterHotKey` gibt `false` zurück. `RegisterHotKey` gibt auch manchmal bei anderen Tasten, wie z. B. der F11-Taste (ohne Modifizierer) einen Fehler zurück. Leider konnte ich nicht herausfinden, was die wirkliche Ursache dafür war.

Hinweis

Die Fehlerauswertung bei einer Rückgabe von `false` beim Aufruf von `RegisterHotkey` und `UnregisterHotkey` beschränkt sich leider scheinbar auf die Auswertung des Fehlercodes, den `Marshal.GetLastWin32Error` zurückgibt. Alle Versuche, über `FormatMessage` (die ich in der Einführung dieses Buchs erläutert habe) eine sinnvolle Fehlermeldung zu erhalten, sind leider gescheitert. Die Dokumentation von `RegisterHotKey` und `UnregisterHotKey` schweigt sich auch leider (wie auch das restliche Internet) über die Bedeutung der Fehlercodes aus. Schade.

Windows-Formulare mit einem System-Hotkey verknüpfen

Mit den API-Funktionen ist es ein Leichtes, eine Anwendung mit einem Hotkey zu versehen. Listing 356 demonstriert dies, indem beim Laden eines Formulars die Tastenkombination SHIFT + STRG + T als Hotkey registriert wird.

```
/* Deklaration der RegisterHotKey-Funktion */
[DllImport("user32.dll")]
[return: MarshalAs(UnmanagedType.Bool)]
static extern bool RegisterHotKey(IntPtr hWnd, int id, uint fsModifiers,
    uint vk);

/* Deklaration der UnregisterHotKey-Funktion */
[DllImport("user32.dll")]
[return: MarshalAs(UnmanagedType.Bool)]
static extern bool UnregisterHotKey(IntPtr hWnd, int id);
```

Listing 356: Beispiel für das Registrieren eines systemweiten Hotkey in einem Formular

```
/* Verwaltet die Modifiziertasten für die Zuweisung eines Hotkey */
private enum Modifiers : uint
{
   /* Die ALT-Taste */
   Alt = 0x0001,

   /* Die STRG-Taste */
   Control = 0x0002,

   /* Die SHIFT-Taste */
   Shift = 0x0004,

   /* Die Windows-Taste */
   Win = 0x0008
}

/* Die ID des Hotkey */
private const int HOTKEY_ID = 42;

/* Registriert den Hotkey beim Starten des Formulars */
private void StartForm_Load(object sender, EventArgs e)
{
   // Hotkey SHIFT + STRG + T zuweisen. Als Id (die für alle in diesem
   // Thread zugewiesenen Hotkeys eindeutig sein muss) wird das Fenster-
   // Handle selbst übergeben
   if (RegisterHotKey(this.Handle, HOTKEY_ID,
      (uint)(Modifiers.Shift | Modifiers.Control), (uint)Keys.T) == false)
   {
      // RegisterHotKey ist fehlgeschlagen:
      // Ausgabe einer Fehlermeldung
      MessageBox.Show("Fehler " + Marshal.GetLastWin32Error() +
         " beim Registrieren des Hotkey", Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}

/* Wertet die WM_HOTKEY-Nachricht aus */
protected override void WndProc(ref Message m)
{
   // Die geerbte Fenster-Methode aufrufen
   base.WndProc(ref m);

   const int WM_HOTKEY = 0x312;
   if (m.Msg == WM_HOTKEY)
   {
```

Listing 356: Beispiel für das Registrieren eines systemweiten Hotkey in einem Formular (Forts.)

```
      // Die WM_HOTKEY-Nachricht wurde an das Fenster gesendet:
      // Das Fenster ggf. wiederherstellen, sichtbar schalten und aktivieren
      if (this.WindowState == FormWindowState.Minimized)
      {
         this.WindowState = FormWindowState.Normal;
      }
      this.Visible = true;
      this.Activate();
   }
}

/* Gibt den Hotkey wieder frei, wenn das Formular geschlossen wird */
private void StartForm_FormClosed(object sender, FormClosedEventArgs e)
{
   // Hotkey wieder freigegben
   if (UnregisterHotKey(this.Handle, HOTKEY_ID) == false)
   {
      // UnregisterHotKey ist fehlgeschlagen:
      MessageBox.Show("Fehler " + Marshal.GetLastWin32Error() +
         " beim Deregistrieren des Hotkey", Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 356: Beispiel für das Registrieren eines systemweiten Hotkey in einem Formular (Forts.)

Das Beispiel registriert den Hotkey SHIFT + STRG + T im `Load`-Ereignis des Formulars. `RegisterHotKey` gibt `true` zurück, wenn die Registrierung erfolgreich war.

Bei einer nicht erfolgreichen Registrierung kann leider nur der Windows-Fehlercode ausgegeben werden, da ich, wie bereits gesagt, nicht herausgefunden habe, wie die hinter dem Fehlercode steckende Fehlermeldung ermittelt werden kann.

In der überschriebenen `WndProc`-Methode wird die Windows-Nachricht `WM_HOTKEY` abgefangen und in diesem Fall das Formular in den Vordergrund geholt.

Beim Entladen des Formulars wird der Hotkey wieder freigegeben.

Eine Komponente für die freie Verwendung von Hotkeys

Um Hotkeys beliebig auswerten zu können, habe ich die Komponente `WindowsHotkey` entwickelt. Die Grundidee dieser Komponente basiert auf einem Beispiel von Alexander Werner, das bei Code Project zu finden ist (*www.codeproject.com/cs/miscctrl/systemhotkey.asp*).

Diese Komponente kann (in beliebiger Anzahl) auf einem Windows-Formular angelegt (oder im Programmcode erzeugt und verwendet) werden um systemweite Hotkeys zu registrieren und die Betätigung der Tastenkombination auszuwerten. Die Eigenschaft `Keys` verwaltet die Tastenkombination. Sobald diese Eigenschaft gesetzt wird und `Enabled` `true` ist (was die Voreinstellung ist), wird der Hotkey in Windows registriert.

Über das Ereignis HotkeyPressed können Sie die Betätigung der Tastenkombination auswerten. Das Ereignis Error wird bei einem Fehler bei der Registrierung oder Deregistrierung aufgerufen (außer bei der Freigabe der WindowsHotkey-Instanz über Dispose). Im Ereignisargument Source erhalten Sie eine Information darüber, ob der Fehler bei der Registrierung oder Deregistrierung aufgetreten ist. Ist dieses Ereignis nicht zugewiesen, wird im Fehlerfall eine Exception geworfen.

```
public class WindowsHotkey : Component
{
   /* Deklaration der RegisterHotKey-Funktion */
   [DllImport("user32.dll")]
   [return: MarshalAs(UnmanagedType.Bool)]
   static extern bool RegisterHotKey(IntPtr hWnd, int id, uint fsModifiers,
       uint vk);

   /* Deklaration der UnregisterHotKey-Funktion */
   [DllImport("user32.dll")]
   [return: MarshalAs(UnmanagedType.Bool)]
   static extern bool UnregisterHotKey(IntPtr hWnd, int id);

   /* Die Modifiziertasten für die Zuweisung eines Hotkey */
   public enum ModifierKeys : uint
   {
      /* Kein Modifizierer */
      None = 0,

      /* Die ALT-Taste */
      Alt = 0x0001,

      /* Die STRG-Taste */
      Control = 0x0002,

      /* Die SHIFT-Taste */
      Shift = 0x0004,

      /* Die Windows-Taste */
      Win = 0x0008
   }

   /* Gibt die Quelle an, die zum Error-Ereignis geführt hat */
   public enum ErrorSource
   {
      /* Fehler während der Registrierung des Hotkey */
      RegisterHotkey,
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey

```
      /* Fehler während der Deregistrierung des Hotkey */
      UnregisterHotkey
   }

   /* Klasse für ein natives (verstecktes) Fenster, das die WM_HOTKEY-
      Nachricht abfängt */
   private class HotkeyWindow : NativeWindow, IDisposable
   {
      /* Referenz auf das Parent-Objekt. Wird im Konstruktor übergeben. */
      public WindowsHotkey parent;

      /* Konstruktor. Erzeugt den Fensterhandle. */
      public HotkeyWindow(WindowsHotkey parent)
      {
         this.parent = parent;
         CreateParams createParams = new CreateParams();
         this.CreateHandle(createParams);
      }

      /* Behandelt die WM_HOTKEY-Nachricht */
      protected override void WndProc(ref Message m)
      {
         // Die geerbte Fenster-Methode aufrufen
         base.WndProc(ref m);

         const int WM_HOTKEY = 0x312;
         if (m.Msg == WM_HOTKEY)
         {
            // Die WM_HOTKEY-Nachricht wurde an das Fenster gesendet:
            // Die OnHotkeyPressed-Methode des Parent aufrufen.
            this.parent.OnHotkeyPressed();
         }
      }

      /* Gibt das Fenster frei */
      public void Dispose()
      {
         if (this.Handle != (IntPtr)0)
         {
            this.DestroyHandle();
         }
      }
   }

   /* Ereignisargumente für das Error-Ereignis */
   public class ErrorEventArgs : EventArgs
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
{
   /* Der Fehlercode */
   public readonly int ErrorCode;

   /* Die Quelle des Fehlers */
   public readonly ErrorSource Source;

   /* Konstruktor */
   internal ErrorEventArgs(int errorCode, ErrorSource source)
   {
      this.ErrorCode = errorCode;
      this.Source = source;
   }
}

/* Das native Windows-Fenster, das als Empfänger der
   WM_HOTKEY-Nachricht verwendet wird */
private HotkeyWindow hotkeyWindow;

/* Gibt an, ob der Hotkey registriert ist. */
private bool isRegistered = false;

/* Verwaltet die ID des Hotkey */
private int hotkeyId = 0;

/* Verwaltet die nächste Hotkey-Id.
   Wird im Konstruktor in das private Feld hotkeyId geschrieben
   und danach hochgezählt. */
private static int nextHotkeyId = 0;

/* Konstruktor. Erzeugt das native Windows-Fenster, das als Empfänger
   der WM_HOTKEY-Nachricht verwendet wird, und definiert die Hotkey-ID. */
public WindowsHotkey()
{
   // Das Fenster erzeugen, das die WM_HOTKEY-Nachricht abfängt
   this.hotkeyWindow = new HotkeyWindow(this);

   // Die Hotkey-Id setzen
   lock (this)
   {
      this.hotkeyId = WindowsHotkey.nextHotkeyId;
      WindowsHotkey.nextHotkeyId++;
   }
}

private Keys keys = Keys.None;
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
/* Die Taste(nkombination), die den Hotkey definiert. Beachten Sie,
   dass Hotkeys nicht auf die F12-Taste (ohne Modifizierer) gelegt
   werden dürfen, da F12 für den System-Debugger reserviert ist. */
public Keys Keys
{
   get
   {
      return this.keys;
   }

   set
   {
      // Wert auf F12 überprüfen
      if (value == Keys.F12)
      {
         throw new Exception("Keys darf nicht auf F12 ohne " +
            "Modifizierer gesetzt werden, da F12 für den " +
            "System-Debugger reserviert ist");
      }

      // Wert übergeben
      this.keys = value;

      // Hotkey registrieren, wenn zurzeit nicht im Design-Modus,
      // der Hotkey aktiviert und wenn Keys gesetzt ist
      if (this.DesignMode == false && this.enabled &&
         this.keys != Keys.None)
      {
         this.RegisterHotkey();
      }
   }
}

private bool enabled = true;
/* Gibt an, ob der Hotkey aktiviert ist */
public bool Enabled
{
   get
   {
      return this.enabled;
   }
   set
   {
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
            // Wert übergeben
            this.enabled = value;

            // Hotkey registrieren, wenn zurzeit nicht im Design-Modus,
            // der Hotkey aktiviert und wenn Keys gesetzt ist
            if (this.DesignMode == false && this.keys != Keys.None)
            {
               if (this.enabled)
               {
                  this.RegisterHotkey();
               }
               else
               {
                  this.UnregisterHotkey(true);
               }
            }
         }
      }

      /* Wird aufgerufen, wenn der Hotkey betätigt wurde */
      public event EventHandler HotkeyPressed;

      /* Wird aufgerufen, wenn beim Registrieren oder Deregistrieren des
         Hotkeys ein Fehler auftritt. Wenn dieses Ereignis zugewiesen ist,
         wird bei einem Fehler keine Exception geworfen. */
      public event EventHandler<ErrorEventArgs> Error;

      /* Registriert die aktuelle gesetzte Tastenkombination als
         Windows-Hotkey */
      private void RegisterHotkey()
      {
         // Überprüfen, ob der Hotkey bereits registriert ist
         if (this.isRegistered)
         {
            // Den alten Hotkey zunächst deregistrieren
            this.UnregisterHotkey(true);
         }

         // Modifizier-Tasten auslesen
         ModifierKeys modifierKeys = ModifierKeys.None;
         Keys keys = this.keys;
         if ((keys & Keys.Control) > 0)
         {
            modifierKeys |= ModifierKeys.Control;
            keys ^= Keys.Control;
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
      }
      if ((keys & Keys.Shift) > 0)
      {
         modifierKeys |= ModifierKeys.Shift;
         keys ^= Keys.Shift;
      }
      if ((keys & Keys.Alt) > 0)
      {
         modifierKeys |= ModifierKeys.Alt;
         keys ^= Keys.Alt;
      }

      // Den neuen Hotkey registrieren
      if (RegisterHotKey(this.hotkeyWindow.Handle, this.hotkeyId,
         (uint)modifierKeys, (uint)keys) == false)
      {
         // Fehler bei der Registrierung des Hotkey:
         // API-Fehler auslesen
         int apiError = Marshal.GetLastWin32Error();

         if (this.Error != null)
         {
            // Das Fehler-Ereignis aufrufen
            this.OnError(new ErrorEventArgs(apiError,
               ErrorSource.RegisterHotkey));
         }
         else
         {
            // Exception mit der Fehlermeldung werfen
            throw new Exception("Fehler " + apiError +
               " beim Registrieren des Hotkey");
         }
      }

      // Hier ist der Hotkey registriert
      this.isRegistered = true;
   }

   /* Hebt die Registrierung des aktuellen Hotkey auf */
   private void UnregisterHotkey(bool handleError)
   {
      if (this.isRegistered)
      {
         if (UnregisterHotKey(this.hotkeyWindow.Handle, this.hotkeyId))
         {
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
            this.isRegistered = false;
         }
         else if (handleError)
         {
            // Fehler beim Aufheben der Registrierung des Hotkey:
            // API-Fehler auslesen
            int apiError = Marshal.GetLastWin32Error();
            if (this.Error != null)
            {
               // Fehler-Ereignis aufrufen
               this.OnError(new ErrorEventArgs(apiError,
                  ErrorSource.UnregisterHotkey));
            }
            else
            {
               // Exception mit der Fehlermeldung werfen
               throw new Exception("Fehler " + apiError +
                  " beim Deregistrieren des Hotkey");
            }
         }
      }
   }

   /* Ruft das HotkeyPressed-Ereignis auf */
   protected void OnHotkeyPressed()
   {
      if (this.HotkeyPressed != null)
      {
         this.HotkeyPressed(this, new EventArgs());
      }
   }

   /* Ruft das Error-Ereignis auf */
   protected void OnError(ErrorEventArgs e)
   {
      if (this.Error != null)
      {
         this.Error(this, e);
      }
   }

   /* Ruft Dispose des Hotkey-Fensters auf, um den Hotkey freizugeben */
   protected override void Dispose(bool disposing)
   {
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

```
      // Hotkey deregistrieren
      if (this.isRegistered)
      {
         this.UnregisterHotkey(false);
      }

      // Dispose des Hotkey-Fensters aufrufen
      this.hotkeyWindow.Dispose();

      // Die geerbte Methode aufrufen
      base.Dispose(disposing);
   }
}
```

Listing 357: Komponente zur Verwendung eines systemweiten Hotkey (Forts.)

Der wesentliche Trick dieser Komponente ist, dass sie ein unsichtbares, natives Windows-Fenster zur Auswertung der `WM_HOTKEY`-Nachricht verwendet. Dieses Fenster ist über die Klasse `HotkeyWindow` definiert und wird im Konstruktor erzeugt.

Um eine eindeutige ID für den Hotkey zu erhalten, verwaltet die `WindowsHotkey`-Klasse das statische Feld `nextHotkeyId`, dessen Wert im Konstruktor ausgelesen, in das private (Instanz-)Feld `hotkeyId` geschrieben und danach hochgezählt wird.

Beim Setzen von `Keys` wird der Hotkey registriert. Falls bereits ein Hotkey registriert ist, wird dieser zuvor deregistriert. Für den Fall, dass beim Registrieren oder Deregistrieren ein Fehler auftritt, wird das `Error`-Ereignis aufgerufen, falls dieses mit einer Ereignismethode verknüpft ist. Ist dieses nicht verknüpft, wird stattdessen eine `Exception` geworfen. `Keys` kann auch auf `Keys.None` gesetzt werden. In diesem Fall wird der Hotkey einfach nur deregistriert. Dasselbe geschieht, wenn `Enabled` auf `false` gesetzt wird.

Die `HotkeyWindow`-Klasse ruft in der `WndProc`-Methode beim Eintritt der `WM_HOTKEY`-Nachricht die `OnHotkeyPressed`-Methode ihres Parent (der `WindowsHotkey`-Instanz) auf. `OnHotkeyPressed` ruft dann das `HotkeyPressed`-Ereignis auf.

Um ein sauberes Zerstören des Hotkey-Fenstersw und die Freigabe des Hotkey zu erreichen, überschreibt die `WindowsHotkey`-Klasse die `Dispose`-Methode. In dieser Methode wird zunächst `UnregisterHotkey` aufgerufen um den Hotkey zu deregistrieren. Da beim Schließen der Anwendung in meinen Tests beim Deregistrieren Fehler auftraten, werden diese einfach ignoriert (der Hotkey wird trotzdem sauber freigegeben). Danach wird noch die `Dispose`-Methode des Hotkey-Fensters aufgerufen, um dieses freizugeben.

Internet

206 Status der Internetverbindung abfragen

Wenn Sie abfragen wollen, ob auf einem Computer eine Internetverbindung besteht, finden Sie leider auch im .NET Framework 3.5 scheinbar keine Möglichkeit. Sie können jedoch die API-Funktionen `InternetGetConnectedState` und `InternetGetConnectedStateEx` aufrufen, um eine Information über den aktuellen Status der Internetverbindung zu erhalten. `InternetGetConnectedStateEx` bietet gegenüber der anderen Funktion den Vorteil, dass diese zusätzlich zum Status auch den Namen der Internetverbindung zurückliefert.

Listing 358 setzt den Aufruf dieser Funktion in der Methode `GetInternetConnectionState` um. `InternetGetConnectedStateEx` liefert einen Wert ungleich Null zurück, wenn zurzeit eine Internetverbindung besteht. Zusätzliche Informationen über den Status der Internetverbindung werden in Form einer `int`-Bitmaske am `out`-Argument `lpdwFlags` zurückgegeben. Die dazu verwendeten Konstanten sind in der Datei *wininit.h* deklariert. Am ersten Argument erwartet diese Funktion deshalb eine `int`-Variable. Am zweiten Argument wird der Name der Internetverbindung in Form eines String zurückgegeben, weswegen hier ein `StringBuilder`-Objekt übergeben wird, das ausreichend groß initialisiert wurde. Das dritte Argument teilt der API-Funktion lediglich mit, wie viele Zeichen in den String geschrieben werden können, das vierte (reservierte) Argument wird zurzeit nicht benutzt.

Listing 358 enthält zunächst die Deklaration der API-Funktion und der benötigten Konstanten. Die Methode `GetInternetConnectionState` gibt eine Instanz der Klasse `InternetConnectionState` zurück, die deswegen ebenfalls iListing 358n deklariert ist. Diese Klasse beinhaltet den Namen und den Online-Status der Internetverbindung und alle verfügbaren Informationen über die Art der Verbindung. `GetInternetConnectionState` erzeugt zunächst eine `InternetConnectionState`-Instanz und ruft dann `InternetGetConnectedStateEx` auf, wobei die Rückgabe (in einen `bool`-Wert konvertiert) gleich in das Feld `Online` des `InternetConnectionState`-Objekts geschrieben wird. Danach werden der Name und die Status-Flags ausgewertet und in die `InternetConnectionState`-Instanz geschrieben.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.Text` und `System.Runtime.InteropServices` importieren.

```
/* Deklaration der API-Funktion InternetGetConnectedStateEx */
[DllImport("wininet.dll")]
private static extern int InternetGetConnectedStateEx(out int lpdwFlags,
   StringBuilder lpszConnectionName, int dwNameLen, int dwReserved);

/* Konstanten für InternetGetConnectedStateEx */
private const int INTERNET_CONNECTION_MODEM = 0x01;
private const int INTERNET_CONNECTION_LAN = 0x02;
private const int INTERNET_CONNECTION_PROXY = 0x04;
```

Listing 358: Methode zur Ermittlung des Internetverbindungsstatus

```
private const int INTERNET_RAS_INSTALLED = 0x10;
private const int INTERNET_CONNECTION_OFFLINE = 0x20;
private const int INTERNET_CONNECTION_CONFIGURED = 0x40;

/* Klasse für die Rückgabe des Internet-Verbindungsstatus */
public class InternetConnectionState
{
   // Der Name der Verbindung
   public string Name;
   // Info, ob die Verbindung online ist
   public bool Online;
   // Info, ob die Verbindung konfiguriert ist
   public bool Configured;
   // Info, ob eine Modem-Verbindung besteht (auch ISDN und DSL)
   public bool ModemConnection;
   // Info, ob die Verbindung über das LAN erfolgt (z.B. bei TDSL)
   public bool Lan;
   // Info, ob die Verbindung über einen Proxy erfolgt
   public bool ProxyConnection;
   // Info, ob RAS installiert ist
   public bool RASInstalled;
   // Info, ob das System im Offline-Modus ist
   public bool Offline;
}

/* Methode zur Ermittlung der Art der aktuellen Internetverbindung */
public static InternetConnectionState GetInternetConnectionState()
{
   // InternetConnectionState-Instanz erzeugen
   InternetConnectionState ics = new InternetConnectionState();

   // Verbindungsstatus abfragen
   StringBuilder icsName = new StringBuilder(1024);
   int flags;
   ics.Online = (InternetGetConnectedStateEx(out flags, icsName,
      1024, 0) != 0);
   ics.Name = icsName.ToString();
   ics.Configured = ((flags & INTERNET_CONNECTION_CONFIGURED) > 0);
   ics.Lan = ((flags & INTERNET_CONNECTION_LAN) > 0);
   ics.ModemConnection = ((flags & INTERNET_CONNECTION_MODEM) > 0);
   ics.Offline = ((flags & INTERNET_CONNECTION_OFFLINE) > 0);
   ics.ProxyConnection = ((flags & INTERNET_CONNECTION_PROXY) > 0);
   ics.RASInstalled = ((flags & INTERNET_RAS_INSTALLED) > 0);
```

Listing 358: Methode zur Ermittlung des Internetverbindungsstatus (Forts.)

```
   // Das InternetConnectionState-Objekt zurückgeben
   return ics;
}
```

Listing 358: Methode zur Ermittlung des Internetverbindungsstatus (Forts.)

Der Test der Methode `GetInternetConnectionState` hat einige logische Dinge, aber auch einige Ungereimtheiten ergeben:

- Das Feld `Online` ist (nach meinen Tests unter Windows XP und Windows 2000) dummerweise eigentlich fast immer `true`. Wenn zurzeit keine Internetverbindung besteht, aber eine LAN-Verbindung vorhanden ist, wird diese als Internetverbindung gewertet, auch wenn gar keine Internetverbindung besteht.
- Das Feld `Offline` sagt nicht aus, dass die Internetverbindung zurzeit nicht besteht, da dieses Feld auch bei einer nicht bestehenden Internetverbindung `false` ist (!?).
- Das Feld `ModemConnection` ist zuverlässig nur dann `true`, wenn zurzeit eine Verbindung über ein Telefonmodem, ein DSL-Modem, ein ISDN-Modem oder eine ISDN-Karte besteht. Dieses Feld ist also das einzige, das eine konkrete Aussage über den Zustand der Internetverbindung bei einer Modem-Verbindung oder einer Verbindung über eine ISDN-Karte macht.
- `ProxyConnection` ist `true`, wenn das System (über den Internet Explorer) so konfiguriert ist, dass ein Proxy-Server verwendet wird. Dazu muss weder eine Internetverbindung bestehen, noch der Proxy-Server ausgeführt werden.
- `Lan` ist `true`, wenn eine Lan-Verbindung besteht und `ModemConnection false` ist. `Lan` ist leider auch `true`, wenn gar keine Internetverbindung besteht oder wenn eine Modem-Internetverbindung offline ist.

Hinweis

Eine zuverlässige Aussage darüber, ob gerade eine Internetverbindung besteht, können Sie also nur dann über das Feld `ModemConnection` treffen, wenn diese auf dem lokalen System über ein Telefonmodem, ein DSL-Modem, ein ISDN-Modem oder eine ISDN-Karte ausgeführt wird. Um sicherzugehen, dass eine Verbindung besteht, sollten Sie vielleicht besser einen Ping an eine zuverlässige Internetadresse (z. B. *www.google.com*) ausführen (siehe Rezept 207).

Listing 359 zeigt eine beispielhafte Anwendung der `GetInternetConnectionState`-Methode.

```
try
{
   // Internetverbindungs-Status abfragen
   InternetConnectionState ics = GetInternetConnectionState();

   // Ergebnis ausgeben
   Console.WriteLine("Name: {0}", ics.Name);
```

Listing 359: Beispielhafte Anwendung der GetInternetConnectionState-Methode

```
   Console.WriteLine("Online: {0}", ics.Online);
   Console.WriteLine("Configured: {0}", ics.Configured);
   Console.WriteLine("Lan: {0}", ics.Lan);
   Console.WriteLine("ModemConnection: {0}", ics.ModemConnection);
   Console.WriteLine("Offline: {0}", ics.Offline);
   Console.WriteLine("ProxyConnection: {0}", ics.ProxyConnection);
   Console.WriteLine("RASInstalled: {0}", ics.RASInstalled);
}
catch (Exception ex)
{
   Console.WriteLine("Fehler bei der Abfrage des Internet-" +
      "Verbindungsstatus: {0}",
     ex.Message);
}
```


Listing 359: Beispielhafte Anwendung der GetInternetConnectionState-Methode (Forts.)

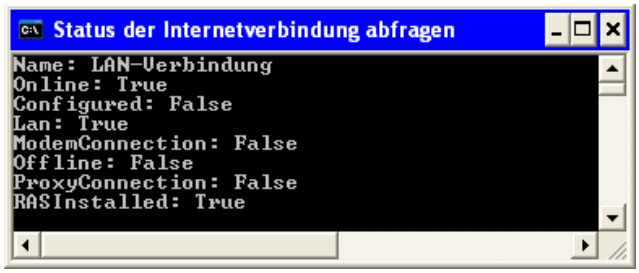


Abbildung 116: Ausführung des Beispielprogramms bei einer bestehenden DSL-Verbindung

207 Pingen – Ermitteln, ob eine Internetverbindung besteht

Wenn Sie ermitteln wollen, ob eine Internetverbindung besteht, reicht es nicht aus, den Status der Internetverbindung abzufragen (siehe Rezept 206), da der Aufruf der API-Funktion `InternetGetConnectedStateEx` leider auch `true` ergibt, wenn keine Internetverbindung, aber eine LAN-Verbindung besteht.

Zur Überprüfung auf eine geöffnete Internetverbindung könnten Sie die API-Funktion `InternetCheckConnection` einsetzen, der Sie eine URL oder `null` am ersten Argument übergeben können, um die Erreichbarkeit der URL oder einfach nur eine vorhandene Internetverbindung zu überprüfen. Leider arbeitete diese Funktion in meinen Tests nicht mit einer DSL-Verbindung oder einer Internetverbindung über einen Proxy-Server (der Aufruf führte zum (über `FormatMessage` ausgelesenen) Fehler »Diese Netzwerkverbindung ist nicht vorhanden«). Laut einigen Newsgroup-Beiträgen können Sie diese Funktion auch nicht einsetzen, wenn die Internetverbindung über das LAN erfolgt. `InternetCheckConnection` hat mit DSL- und LAN-Verbindungen wahrscheinlich ähnliche Probleme wie `InternetGetConnectedStateEx`.

Die sicherste Möglichkeit, eine offene Internetverbindung abzufragen, ist, an eine bekannte und zuverlässige IP-Adresse (wie z. B. 216.239.59.147 für *www.google.com*) einen oder mehrere *Pings* abzusetzen. Ein Ping (Packet Internet Groper = Paket-Internet-(Ab)Taster) ist eine

Technik, bei der ein spezielles IP-Paket an den Port 7 eines Host gesendet wird. Jeder Host sollte die von einem Ping aufgerufene ICMP[1]-Echo-Funktion (definiert in RFC 792) implementieren. Der Empfänger sendet ein spezielles IP-Paket zurück, wenn er erreichbar ist. Da jedes Paket mit einer eindeutigen Sequenznummer und einer Checksumme versehen wird, kann der Sender beim mehrfachen Pingen erkennen, wie viele Pakete auf dem Weg verloren gegangen sind, wie viele Pakete auf dem Weg zerstört wurden und wie viel Zeit das Senden und Empfangen in Anspruch genommen hat.

Hinweis

Zur Überprüfung einer offenen Internetverbindung sollten Sie explizit die IP-Adresse verwenden und nicht den Domänennamen. Der Grund dafür ist, dass ein Ping an einen Domänennamen eine DNS-Abfrage verursacht, die bei Systemen mit einer automatischen Wählverbindung dazu führt, dass die Verbindung geöffnet wird.

Einen Ping können Sie über die gleichnamige Klasse aus dem Namensraum `System.Net.NetworkInformation` ausführen. Dazu erzeugen Sie eine Instanz und rufen die `Send`-Methode auf. `Send` gibt eine Instanz der `PingReply`-Klasse zurück, die in ihrer Eigenschaft `Status` sehr detaillierte Angaben über den Status des Ping gibt. Die für uns wichtigsten habe ich in Tabelle 24 aufgeführt.

PingReply-Status	Bedeutung
`IPStatus.Sucess`	Der Ping war erfolgreich
`IPStatus.TimedOut`	Das erwartete ICMP-Echo wurde nicht in der über den Timeout definierten Zeit empfangen
`IPStatus.TimeExceeded`	Beim Routen der ICMP-Echo-Anforderung wurde die TTL (Time To Live) an einem Router 0, weswegen dieser die Anforderung verwarf
`IPStatus.TtlExpired`	wie `IPStatus.TimeExceeded`
`IPStatus.DestinationHostUnreachable`	Der Zielhost ist nicht erreichbar. Wahrscheinlich besteht keine Internetverbindung.
`IPStatus.HardwareError`	Hardwarefehler. Wahrscheinlich wurde die LAN-Leitung entfernt oder der Router ist ausgeschaltet.

Tabelle 24: Die wichtigsten Werte der IPStatus-Aufzählung

Zum Pingen können Sie der `Send`-Methode am ersten Argument die Adresse des Webservers übergeben und am zweiten Argument einen Timeout in Millisekunden. Prinzipiell reicht dies aus, um das Vorhandensein eines Webservers abzufragen. Weiteren Varianten dieser Methode können Sie zusätzlich noch ein Byte-Array mit Daten übergeben, die gesendet werden sollen, und spezielle Ping-Optionen, die u. a. den TTL-Wert definieren.

Listing 360 verwendet diese Methode, um das Vorhandensein des Webservers von Google abzufragen. Das Beispiel erfordert den Import der Namensräume `System` und `System.Net.NetworkInformation`.

1. ICMP = Internet Control Message Protocol

```
Ping ping = new Ping();

string address = "216.239.59.147";
int timeout = 5000;
PingReply pingReply = ping.Send(address, timeout);

switch (pingReply.Status)
{
   case IPStatus.Success:
      Console.WriteLine("Ping an {0} war erfolgreich. " +
         "Benötigte Zeit: {1} Millisekunden",
         address, pingReply.RoundtripTime);
      break;

   case IPStatus.TimedOut:
      Console.WriteLine("Timeout beim Ping an {0}", address);
      break;

   case IPStatus.TimeExceeded:
      Console.WriteLine("Der TTL-Wert erreichte 0 beim Ping an {0}",
         address);
      break;

   case IPStatus.DestinationHostUnreachable:
      Console.WriteLine("Google ist nicht erreichbar. " +
         "Wahrscheinlich besteht keine Internetverbindung.");
      break;

   case IPStatus.HardwareError:
      Console.WriteLine("Hardware-Fehler. Wahrscheinlich wurde die " +
         "LAN-Leitung entfernt oder der Router ist ausgeschaltet.");
      break;

   default:
      Console.WriteLine("Status beim Ping an {0}: {1}",
      address, pingReply.Status);
      break;
}
```

Listing 360: Prüfen, ob eine Internetverbindung nach www.google.com (216.239.59.147) möglich ist

208 TCP- und ICMP-Prüfsumme berechnen

Wenn Sie TCP- oder ICMP-Daten roh versenden oder empfangen, müssen Sie u. U. die Prüfsumme der Daten berechnen. Diese Prüfsumme wird mit den einzelnen Wörtern der TCP-/ICMP-Daten berechnet, bei TCP in das neunte Wort und bei ICMP in das zweite Wort geschrieben.

Die Methode GetTCPCheckSum in Listing 361 nimmt diese Berechnung vor. GetTCPCheckSum bekommt ein Byte-Array mit den TCP- oder ICMP-Daten übergeben. Bei der ersten Berechnung einer Prüfsumme müssen die Bytes, die die Prüfsumme aufnehmen, 0 speichern. Wenn Sie die Prüfsumme mit Daten berechnen, deren Header bereits eine Prüfsumme speichert, muss (aufgrund des Einer-Komplements) 0 resultieren. Daran können Sie erkennen, dass die Prüfsumme empfangener Daten in Ordnung ist und die Daten somit nicht verändert wurden.

Ich habe GetTCPCheckSum übrigens basierend auf einigen Beispielen im Internet entwickelt. Das maßgebliche Beispiel war die C-Implementierung des Ping-Befehls des Ping-Erfinders Mike Muus (*www.ping127001.com/pingpage/ping.html*).

Zum Kompilieren dieser Methode müssen Sie den Namensraum System einbinden.

```
public static ushort GetTCPCheckSum(byte[] tcpData)
{
   // Wenn das byte-Array am Ende kein ganzes Wort mehr ergibt, muss ein 0-
   // Byte angehängt werden
   byte[] buffer;
   if ((tcpData.Length % 2) > 0)
   {
      buffer = new byte[tcpData.Length + 1];
      tcpData.CopyTo(buffer, 0);
      buffer[buffer.Length - 1] = 0;
   }
   else
   {
      buffer = new byte[tcpData.Length];
      tcpData.CopyTo(buffer, 0);
   }

   // Die einzelnen Wörter addieren
   int checkSum = 0;
   for (int i = 0; i < buffer.Length; i += 2)
   {
      byte lowByte = buffer[i+1];
      byte highByte = buffer[i];
      ushort wordValue = highByte;
      wordValue = (ushort)((wordValue << 8) + lowByte);
      checkSum += wordValue;
   }

   // Nur die niedrigen 16 Bits addiert mit den hohen 16 Bits auslesen
   checkSum = (checkSum >> 16) + (checkSum & 0xFFFF);
   checkSum += (checkSum >> 16);
```

Listing 361: Methode zur Berechnung einer TCP/ICMP-Prüfsumme

```
    // Einerkomplement bilden
    return (ushort)(~checkSum);
}
```

Listing 361: Methode zur Berechnung einer TCP/ICMP-Prüfsumme (Forts.)

209 Internetverbindung öffnen und schließen

Über die API-Funktion `InternetAutodial` können Sie die Standard-Internet-Wählverbindung des Rechners, auf dem das Programm ausgeführt wird, öffnen und über `InternetAutodialHangup` wieder schließen.

Hinweis

`InternetAutodial` öffnet leider nur Internet-Wählverbindungen (über ein Modem oder eine ISDN-Karte). DSL-Verbindungen (die keine Wählverbindungen sind) können mit `InternetAutodial` nicht geöffnet werden. Ähnliches gilt für `InternetAutodialHangup`, die eine DSL-Verbindung nicht schließt. Eine Lösung zum Öffnen und Schließen einer DSL-Verbindung habe ich bisher nicht gefunden.

`InternetAutodial` erwartet am ersten Argument eine `int`-Konstante, die Informationen darüber enthält, wie die Verbindung geöffnet werden soll:

- `INTERNET_AUTODIAL_FORCE_ONLINE` bewirkt, dass eine Online-Verbindung erzeugt wird,
- `INTERNET_AUTODIAL_FORCE_UNATTENDED` führt dazu, dass die Verbindung im Hintergrund geöffnet wird, ohne dass der Anwender im Einwahldialog den Wählen-Schalter betätigen muss,
- `INTERNET_AUTODIAL_FAILIFSECURITYCHECK` bewirkt, dass `InternetAutodial` fehlschlägt, wenn die Drucker- und Datei-Freigabe deaktiviert ist.

Sie können an diesem Argument nur eine der Konstanten angeben (das Argument ist keine Bitmaske). Am zweiten Argument können Sie den Handle eines Formulars übergeben, das dann als Parent für den Einwahl-Dialog verwendet wird. Wenn Sie kein Parent-Formular verwenden (wollen), können Sie hier auch 0 übergeben.

Die Funktion `InternetAutodialHangup`, der Sie am ersten, für spätere Zwecke reservierten Argument 0 übergeben, schließt die Standard-Internetverbindung.

Beide Funktionen geben `true` zurück, wenn die Ausführung erfolgreich war. Leider können Sie damit nicht unbedingt eine Aussage treffen, dass die Verbindung auch geöffnet ist: Ist die Standard-Netzwerkverbindung eine DSL-Verbindung, geben beide Funktionen ebenfalls `true` zurück, obwohl die Verbindung weder geöffnet noch geschlossen wird. Sie sollten also nach dem Öffnen gegebenenfalls überprüfen, ob eine Verbindung besteht (siehe Rezept 207).

Die im Folgenden beschriebenen Methoden `OpenLocalConnection` und `CloseLocalConnection` übernehmen das Öffnen und Schließen der Internetverbindung. Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System`, `System.Text` und `System.Runtime.InteropServices` importieren und die notwendigen API-Funktionen und -Konstanten deklarieren. Um eventuelle (leider undokumentierte) Fehler bei der Ausführung der Funktionen auswerten zu

können, habe ich die Funktion FormatMessage mit aufgenommen. Da die Fehlermeldungen von InternetAutodial und InternetAutodialHangup nicht in der Windows-System-Meldungstabelle, sondern in der Datei *wininet.dll* gespeichert sind, ist die Funktion GetModuleHandle ebenfalls deklariert.

```
/* Deklaration der benötigten API-Funktionen  */
[DllImport("wininet.dll", SetLastError=true)]
public static extern int InternetAutodial(int dwFlags,
   IntPtr hwndParent);

[DllImport("wininet.dll", SetLastError=true)]
public static extern int InternetAutodialHangup(int dwReserved);

[DllImport("Kernel32.dll")]
private static extern int FormatMessage(int dwFlags, IntPtr lpSource,
   int dwMessageId, int dwLanguageId,
   System.Text.StringBuilder lpBuffer, int nSize, string [] Arguments);

[DllImport("kernel32.dll")]
private static extern IntPtr GetModuleHandle(string lpFileName);

/* Konstanten für InternetAutoDial */
private const int INTERNET_AUTODIAL_FORCE_ONLINE = 1;
private const int INTERNET_AUTODIAL_FORCE_UNATTENDED = 2;
private const int INTERNET_AUTODIAL_FAILIFSECURITYCHECK = 4;

/* Konstante für FormatMessage */
private const int FORMAT_MESSAGE_FROM_HMODULE = 0x0800;
```

Listing 362: API-Deklarationen für InternetAutodial und InternetAutodialHangup

Die Methode OpenLocalConnection in Listing 363 öffnet eine Wählverbindung zum Internet. Am Argument unattendet wird definiert, ob das Öffnen unbeaufsichtigt erfolgen soll (also ohne dass der Anwender den Wählen-Schalter des Verbindungsdialogs betätigen muss). Das Argument hwndParent kann mit dem Windows-Handle eines Formulars belegt werden, das als Parent-Fenster des Einwahl-Dialogs verwendet werden soll.

OpenLocalConnection definiert zunächst die zu übergebenden Flags und ruft dann InternetAutodial auf. Wenn die Funktion 0 zurückliefert, liest OpenLocalConnection den letzten API-Fehlercode aus und versucht, damit aus der Datei *wininet.dll* eine Fehlerbeschreibung auszulesen.

Hinweis

Leider führten meine Versuche nicht dazu, dass eine Beschreibung ausgelesen werden konnte. Der durch einen Abbruch des Wähl-Dialogs erzeugte Fehlercode 631 ist nicht mit einer Meldung verknüpft (die Fehlercodes in der Meldungstabelle der *wininet.dll* beginnen auch erst bei 12.000, was ich über das Test-Programm »Fehlercodes ermitteln« herausgefunden habe, das Sie im Ordner *Beispiele/Tools* auf der Buch-CD finden).

```
public static void OpenLocalConnection(bool unattended, int hwndParent)
{
   // Flags definieren
   int flags = 0;
   if (unattended)
   {
      flags = INTERNET_AUTODIAL_FORCE_UNATTENDED;
   }
   else
   {
      flags = INTERNET_AUTODIAL_FORCE_ONLINE;
   }

   // Verbindung wählen
   if (InternetAutodial(flags, (IntPtr)hwndParent) == 0)
   {
      // Fehler bei der Ausführung
      int apiError = Marshal.GetLastWin32Error();

      // Den Text des Fehlers aus der Datei wininet.dll auslesen
      // und damit eine Ausnahme werfen
      IntPtr hModule = GetModuleHandle("wininet.dll");
      StringBuilder message = new StringBuilder(1024);
      if (FormatMessage(FORMAT_MESSAGE_FROM_HMODULE, hModule,
         apiError, 0, message, 1024, null) > 0)
      {
         throw new Exception(message.ToString());
      }
      else
      {
         throw new Exception("API-Fehler " + apiError);
      }
   }
}
```

Listing 363: Methode zum Öffnen einer lokalen Internetverbindung

Die Methode `CloseLocalConnection` ruft `InternetAutodialHangup` auf um die aktuelle Verbindung zu schließen. Fehler bei der Ausführung werden auf dieselbe Weise ausgewertet wie bei `OpenLocalConnection`.

```
public static void CloseLocalConnection()
{
   // Verbindung schließen
   if (InternetAutodialHangup(0) == 0)
```

Listing 364: Methode zum Schließen einer lokalen Internetverbindung

```
   {
      // Fehler bei der Ausführung
      int apiError = Marshal.GetLastWin32Error();

      // Den Text des Fehlers aus der Datei wininet.dll auslesen
      // und damit eine Ausnahme werfen
      IntPtr hModule = GetModuleHandle("wininet.dll");
      StringBuilder message = new StringBuilder(1024);
      if (FormatMessage(FORMAT_MESSAGE_FROM_HMODULE, hModule, apiError,
         0, message, 1024, null) > 0)
      {
         throw new Exception(message.ToString());
      }
      else
      {
         throw new Exception("API-Fehler " + apiError);
      }
   }
}
```

Listing 364: Methode zum Schließen einer lokalen Internetverbindung (Forts.)

Listing 365 zeigt eine Beispiel-Anwendung der Methode `OpenLocalConnection`, bei der das Öffnen nicht unbeaufsichtigt erfolgt.

```
try
{
   OpenLocalConnection(false, 0);

   Console.WriteLine("Internetverbindung ist geöffnet");
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 365: Anwendung der Methode OpenLocalConnection

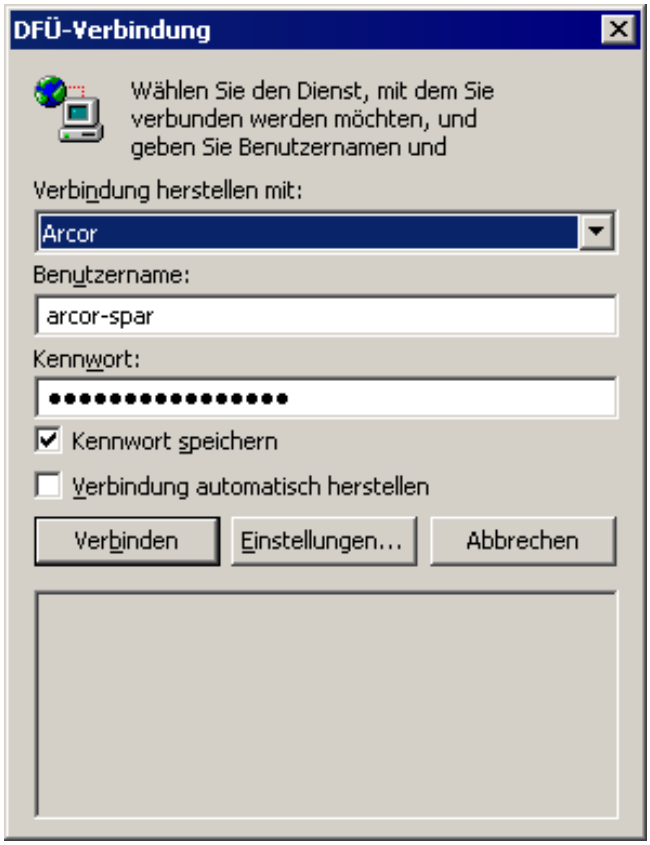


Abbildung 117: Der DFÜ-Einwahldialog, der nach dem Aufruf von OpenLocalConnection geöffnet wurde

210 Die IP-Adressen des Computers herausfinden

Normalerweise besitzt ein Computer mindestens eine, meist aber sogar mehrere IP-Adressen. Eine Adresse liegt häufig im Bereich des lokalen Netzwerks und ist der Netzwerkkarte zugeordnet, die für die lokale Netzwerkverbindung verwendet wird. Eine weitere Adresse ist häufig der aktuellen Internetverbindung zugeordnet, wenn der Computer eine solche direkt (und nicht über das Netzwerk) aufbaut.

Sie können die IP-Adressen eines Computers über DNS herausfinden. Die Methode `GetHostByName` der `Dns`-Klasse aus dem Namensraum `System.Net`, der Sie den Namen des Computers übergeben können, liefert ein `IPHostEntry`-Objekt zurück, das neben dem Host-Namen und Aliasnamen für den Host in der Eigenschaft `AddressList` ein Array von `IPAddress`-Objekten verwaltet. Über die Eigenschaft `Address` eines solchen Objekts erhalten Sie die jeweilige IP-Adresse, allerdings als `long`-Wert. Da die übliche Darstellung einer IP-Adresse die vier Bytes des `long`-Werts einzeln und durch Punkte getrennt darstellt, sollten Sie stattdessen die `ToString`-Methode aufrufen, die die IP-Adresse in der üblichen Form zurückgibt. `GetHostByName` können Sie leider nicht den Namen *localhost* oder die IP-Adresse 127.0.0.1 übergeben um den lokalen Computer anzusprechen, in diesem Fall resultiert lediglich die Adresse 127.0.0.1. Sie

müssen den DNS-Namen des Computers herausfinden und dieser Methode übergeben, was aber über `Dns.GetHostEntry` sehr einfach ist.

Das Beispiel erfordert den Import der Namensräume `System` und `System.Net`.

```
IPAddress[] addressList = Dns.GetHostEntry(Dns.GetHostEntry()).AddressList;
for (int i = 0; i < addressList.Length; i ++)
{
   string ipAddress = addressList[i].ToString();
   Console.WriteLine(ipAddress);
}
```

Listing 366: Die IP-Adressen des lokalen Computers über DNS auslesen

Diese Variante funktioniert nur dann, wenn DNS auf dem lokalen System korrekt konfiguriert und nicht deaktiviert ist. Das ist zwar bei den meisten Systemen der Fall, Sie können dies aber nicht voraussetzen. Deshalb sollten Sie die andere Lösung einsetzen, die die IP-Adressen über WMI abfragt. Diese Lösung geht alle Netzwerkadapter des Systems über die WMI-Klasse `Win32_NetworkAdapterConfiguration` durch, fragt ab, ob der Adapter IP unterstützt, und liest die den Adapter zugeordneten IP-Adressen aus.

Die Methode `GetLocalIPAddresses` in Listing 367 geht auf diese Weise vor. Um die Verarbeitung der Adressen zu erleichtern gibt `GetLocalIPAddresses` eine Auflistung vom Typ `List` zurück, die mit dem Typ `string` typisiert ist.

Zum Kompilieren der Methode müssen Sie die Assembly *System.Management.dll* referenzieren und die Namensräume `System`, `System.Collections.Generic` und `System.Management` importieren.

```
public static List<string> GetLocalIPAddresses()
{
   List<string> ipAddressList = new List<string>();

   try
   {
      // Alle Netzwerkadapter des Systems ermitteln und durchgehen
      ManagementClass mc = new ManagementClass(
         "Win32_NetworkAdapterConfiguration");
      ManagementObjectCollection moc = mc.GetInstances();
      foreach (ManagementObject mo in moc)
      {
         if ((bool)mo["IpEnabled"])
         {
            // Wenn der Adapter das IP-Protokoll unterstützt:
            // Adressen auslesen
            string[] ipAddresses = (string[])mo["IPAddress"];
```

Listing 367: Methode zum Auslesen der lokalen IP-Adressen

```
            foreach (string ipAddress in ipAddresses)
            {
               ipAddressList.Add(ipAddress);
            }
         }
      }
   }
   catch {}

   return ipAddressList;
}
```

Listing 367: Methode zum Auslesen der lokalen IP-Adressen (Forts.)

Abbildung 118 zeigt die Ausgabe einer Konsolenanwendung, die die IP-Adressen einmal über DNS und zum anderen über `GetLocalIPAddresses` auf meinem System aufgerufen und die Adressen jeweils ausgegeben hat.

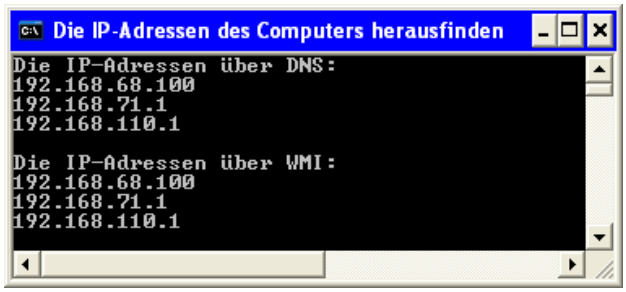


Abbildung 118: Auslesen der lokalen IP-Adressen

Das Problem, herauszufinden, welche der ermittelten Adressen nun die Adresse der Internetverbindung ist, konnte ich bisher leider nicht lösen. Ein Lösungsansatz ist, die IP-Adresse mit der Subnetzmaske des Providers zu vergleichen, sofern diese sich von der lokalen Subnetzmaske unterscheidet.

211 E-Mails über einen SMTP-Server versenden

Das Versenden von E-Mails ist nicht nur interessant für die Programmierung von eigenen E-Mail-Clients (von denen es ja eigentlich bereits mehr als genug gibt). E-Mails sind auch sinnvoll für Anwendungen, die bestimmte Personen (in einem Unternehmen) über einen neu eingetretenen Zustand informieren müssen. In einem Projekt-Planungssystem ist es z. B. sinnvoll, alle Projekt-Mitarbeiter in Kenntnis zu setzen, nachdem wesentliche Daten wie der Einführungstermin vom Projektleiter geändert wurden. Eine elegante Programmlösung versendet entsprechende E-Mails in solchen Fällen automatisch und nimmt den Mitarbeitern so einiges an Arbeit ab.

Sofern ein SMTP-Server zur Verfügung steht, können Sie zum Senden von E-Mails die Klasse `SmtpClient` aus dem Namensraum `System.Net.Mail` verwenden. Falls kein SMTP-Server, sondern ein Exchange-Server zur Verfügung steht, finden Sie eine Lösung übrigens im Rezept 212.

Hinweis

Viele SMTP-Server erlauben die Weitergabe (englisch: *Relay*) von E-Mails nur unter bestimmten Voraussetzungen. Der lokale SMTP-Server der Internet-Informationsdienste von Windows XP ist zum Beispiel per Voreinstellung so eingestellt, dass er E-Mails nur für Clients, die sich erfolgreich gegenüber dem Server authentifizieren (was auch immer damit gemeint ist), weitergibt. Wenn Sie die Konfiguration dieses SMTP-Servers nicht geändert haben, erhalten Sie beim Versuch, eine E-Mail zu senden, eine `SmtpFailedRecipientException` mit der Meldung »Unable to relay for xyz« (wobei XYZ eine Empfänger-E-Mail-Adresse ist). In diesem Fall müssen Sie die Weiterleitung für die IP-Adressen, die auf den SMTP-Server zugreifen, explizit erlauben.

Die Einstellung der IP-Adressen, für die der Windows-XP-SMTP-Server E-Mails weiterleiten darf, erreichen Sie über die Verwaltung der Internet-Informationsdienste in der Systemsteuerung (SYSTEMSTEUERUNG / VERWALTUNG / INTERNET-INFORMATIONSDIENSTE). Öffnen Sie den Knoten für den lokalen Computer und danach die Eigenschaften des SMTP-Servers. Im Register ZUGRIFF können Sie über den Schalter WEITERGABE die Weitergabeeinschränkungen bearbeiten. Erfolgt der Zugriff lediglich über lokale Clients, müssen Sie in der Regel der IP-Adresse 127.0.0.1 (localhost) Zugriff gewähren (Abbildung 119).

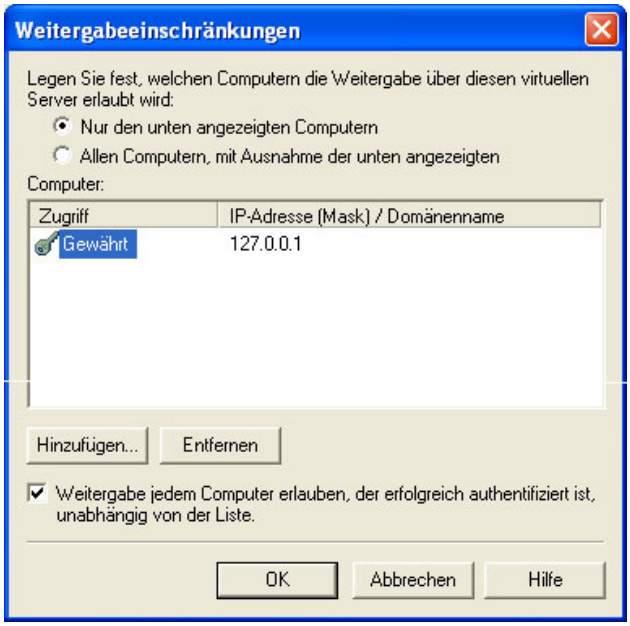


Abbildung 119: Weitergabeeinschränkungen mit Erlaubnis für die IP-Adresse 127.0.0.1 für den lokalen SMTP-Server von Windows XP

Hinweis

Bei Ihren Versuchen, über einen SMTP-Server E-Mails zu versenden, sollten Sie ebenfalls beachten, dass viele Server besondere Techniken verwenden, um den Missbrauch durch Spammer einzuschränken. *mail.isis.de* überprüft z. B., ob die IP-Adresse des Senders zum Netz von Isis gehört. Ist dies nicht der Fall, resultiert der Versuch, eine E-Mail zu senden, in dem Fehler »Relay access denied«. Der SMTP-Server von GMX (*mail.gmx.de*) erfordert eine Authentifizierung und überprüft, ob die in der E-Mail angegebene Sender-Adresse eine Adresse des eingeloggten Benutzers ist. Im anderen Fall resultiert der Fehler »Sender address does not belong to logged in user«.

Zum Senden einer E-Mail erzeugen Sie zunächst eine Instanz der `SmtpClient`-Klasse. Dem Konstruktor können Sie die Adresse des SMTP-Servers übergeben (sofern Sie die Konfiguration nicht in der Anwendungs-Konfigurationsdatei vornehmen und nicht die Pickup-Auslieferer-Methode über den IIS verwenden wollen, die ich weiter unten beschreibe):


```
// SmtpClient erzeugen
string smtpHost = "mail.galaxy.com";
SmtpClient smtpClient = new SmtpClient(smtpHost);
```

Wenn Sie den lokalen SMTP-Server angeben wollen (der ein Teil der Windows-Internet-Informationsdienste ist), geben Sie *localhost* an.

Da viele SMTP-Server eine Authentifizierung verlangen, können Sie die dazu notwendigen Informationen über die Eigenschaft `Credentials` übergeben. Dazu schreiben Sie die Referenz auf ein neues `NetworkCredential`-Objekt in dieser Eigenschaft. Im Konstruktor übergeben Sie diesem Objekt am ersten Argument den Benutzernamen und am zweiten das Passwort:

```
// Authentifizierungs-Informationen für SMTP-Server,
// die eine Authentifizierung verlangen
string userName = "jb";
string password = "galaxy";
smtpClient.Credentials = new NetworkCredential(userName, password);

// Authentifizierungs-Informationen für den SMTP-Server von
// Windows XP, wenn dieser so eingestellt ist, dass die
// Clients sich über die Windows-Authentifizierung anmelden müssen
//smtpClient.Credentials = CredentialCache.DefaultNetworkCredentials;
```

Die Klasse `SmtpClient` ermöglicht verschiedene Ausliefer-Methoden, die Sie über die Eigenschaft `DeliveryMethod` einstellen können. Die Voreinstellung `SmtpDeliveryMethod.Network` sendet die E-Mail an den SMTP-Server an der über die `Host`-Eigenschaft des `SmtpClient`-Objekts angegebenen Adresse. Über `SmtpDeliveryMethod.PickupDirectoryFromIis` können Sie festlegen, dass die E-Mail im Pickup-Verzeichnis des IIS (per Voreinstellung *C:\Inetpub\mailroot\Pickup*) abgelegt wird. `SmtpDeliveryMethod.SpecifiedPickupDirectory` erlaubt die Eingabe eines eigenen Pickup-Verzeichnisses über die `PickupDirectoryLocation`-Eigenschaft des `SmtpClient`-Objekts. Bei einer Pickup-Auslieferung wird kein SMTP-Server verwendet. Hierbei kümmert sich der IIS um die Auslieferung der E-Mail.

```
// Ausliefer-Methode festlegen (Network ist Default)
smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
```

Alternativ können Sie den SMTP-Server, die Authentifizierungsinformationen, die Absender-Adresse, die Ausliefer-Methode und weitere Grundeinstellungen auch in der Anwendungs-Konfigurationsdatei angeben:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.net>
    <mailSettings>
      <smtp from="jb@galaxy.com" deliveryMethod="Network">
        <network host="mail.galaxy.com" userName="jb" password="galaxy" />
      </smtp>
    </mailSettings>
  </system.net>
</configuration>
```

Listing 368: Einstellung der SMTP-Zugangsdaten in der Anwendungs-Konfigurationsdatei

In diesem Fall können Sie eine Instanz der `SmtpClient`-Klasse über den parameterlosen Konstruktor erzeugen und brauchen den SMTP-Server, die Authentifizierungs-Informationen, den Absender und die Ausliefer-Methode nicht anzugeben (was Sie aber trotzdem natürlich können).

Über die Methode `Send` können Sie E-Mails versenden. Der ersten Variante dieser Methode übergeben Sie vier Strings, wobei der erste den Sender, der zweite die Empfänger, der dritte den Betreff und der letzte den Text beinhaltet. Die zweite `Send`-Variante ist allerdings wesentlich flexibler. Dieser Variante übergeben Sie ein `MailMessage`-Objekt, über das Sie unter anderem auch Anhänge versenden können. Listing 370 zeigt, wie Sie ein solches Objekt erzeugen. Ich denke, die meisten Eigenschaften der `MailMessage`-Klasse erklären sich von selbst.

```
// Nachricht erzeugen und initialisieren
MailMessage message = new MailMessage();
message.From = new MailAddress("jb@galaxy.com", "Jürgen Bayer");
message.To.Add("zaphod@galaxy.com");
message.CC.Add("ford@galaxy.com");
message.Bcc.Add("trillian@galaxy.com");
message.Subject = "Party";
message.IsBodyHtml = true;
message.Body = "Hallo Zaphod,<br><br>" +
   "Lust auf eine Party im <i>Restaurant am Ende der Galaxis</i>?";
```

Listing 369: Erzeugen und Initialisieren eines MailMessage-Objekts

Im Beispiel setze ich die Eigenschaft `IsBodyHtml` des `MailMessage`-Objekts auf `true` und schreibe einen HTML-Text in den Body, um die E-Mail im HTML-Format zu versenden.

Um Sonderzeichen wie unsere Umlaute korrekt übertragen zu können, ist wichtig, dass Sie die Codierung des Body auf UTF-8 einstellen (UTF-8 ist eine Unicode-Codierung, die zum einen

alle Zeichen des Unicode-Zeichensatzes beinhaltet und zum anderen von allen Routern und E-Mail-Servern im Internet unterstützt wird):

```
message.BodyEncoding = Encoding.UTF8;
```

Die Priorität der Nachricht können Sie über die `Priority`-Eigenschaft festlegen:

```
message.Priority = MailPriority.High;
```

Neben der Priorität sind in der Praxis das Anfordern einer Übertragungs- und das Anfordern einer Lesebestätigung wichtig.

Hinweis

Übertragungsbestätigungen können erfolgen, wenn eine E-Mail in dem Mailserver eingegangen ist, der das Mailkonto des Empfängers verwaltet, wenn die E-Mail von keinem Mailserver entgegengenommen wird und/oder wenn der Empfang verzögert wird.

Eine Lesebestätigung wird vom E-Mail-Client gesendet, wenn dieser die E-Mail vom Mail-Server abgerufen und der Anwender die empfangene E-Mail geöffnet hat. Viele E-Mail-Clients senden angeforderte Lesebestätigungen automatisch, bei anderen (wie z. B. bei Outlook) kann der Anwender einstellen, ob Lesebestätigungen überhaupt, nur nach einer entsprechenden Nachfrage oder automatisch gesendet werden. In einigen Netzwerken wird das Senden von Lesebestätigungen auch verhindert. Sie können also nicht sicher sein, dass Sie eine Lesebestätigung auch erhalten.

Übertragungsbestätigungen werden per Voreinstellung beim Senden von E-Mails nicht angefordert. Sie müssen diese gegebenenfalls über die Eigenschaft `DeliveryNotificationOptions` des `MailMessage`-Objekts einstellen, indem Sie diese auf eine Kombination der Werte der `DeliveryNotificationOptions`-Aufzählung setzen. Das folgende Beispiel fordert eine Bestätigung für die erfolgreiche, die nicht erfolgreiche und die verzögerte Auslieferung der E-Mail an:

```
message.DeliveryNotificationOptions = DeliveryNotificationOptions.OnSuccess |
   DeliveryNotificationOptions.OnFailure | DeliveryNotificationOptions.Delay;
```

Das Anfordern einer Lesebestätigung ist leider nicht in die `MailMessage`-Klasse als Eigenschaft integriert. Um eine Bestätigung für das Lesen einer E-Mail einzuholen, müssen Sie den Nachrichten-Kopfeinträgen einen Header hinzufügen. Outlook verwendet dazu den Header »Return-Receipt-To«. Der Wert des Header-Eintrags ist die E-Mail-Adresse, die die Bestätigung empfangen soll (also in der Regel die Sender-Adresse). Leider schreibt die `SmtpClient`-Klasse die Header-Namen klein in die SMTP-Nachricht. Outlook zumindest scheint mit dem kleingeschriebenen »return-receipt-to«-Header aber Probleme zu haben und erkennt nicht, dass eine Lesebestätigung angefordert wurde. Als Alternative können Sie den Header »Disposition-Notification-To« verwenden, den Outlook auch kleingeschrieben versteht. Beide Header sind kein Teil des SMTP-Standards (RFC 822, siehe *www.ietf.org/rfc/rfc0822.txt*). Zur Sicherheit füge ich beide Header an (und hoffe, dass die gängigen E-Mail-Clients damit keine Probleme haben):

```
message.Headers.Add("Return-Receipt-To", message.From.Address)
message.Headers.Add("Disposition-Notification-To", message.From.Address);
```

Über die `Add`-Methode der `Attachments`-Eigenschaft können Sie der Nachricht Dateien anhängen. Dazu übergeben Sie eine neue Instanz der `MailAttachment`-Klasse, der Sie im Konstruktor den Dateinamen übergeben.

```
string fileName = Path.Combine(
   Path.GetDirectoryName(Assembly.GetEntryAssembly().Location),
   "dontpanic.gif");
message.Attachments.Add(new Attachment(fileName));
```

Schließlich können Sie die E-Mail über die `Send`-Methode der `SmtpClient`-Instanz versenden. Da beim Senden von E-Mails auch Exceptions vorkommen, deren aussagekräftige Meldungen sich in einer der inneren Exceptions verstecken, habe ich dem Beispiel die Methode `GetExceptionMessages` aus dem Rezept 45 hinzugefügt, die auch die Nachrichten der inneren Exceptions ermittelt. Diese Methode habe ich allerdings nicht in das Listing 370 aufgenommen.

Das Beispiel erfordert den Import der Namensräume `System`, `System.IO`, `System.Net`, `System.Net.Mail`, `System.Text`, `System.Windows.Forms`, `System.Text.RegularExpressions` und `System.Reflection`.


```
try
{
   smtpClient.Send(message);
   MessageBox.Show("E-Mail erfolgreich versendet", Application.ProductName,
      MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (SmtpFailedRecipientException ex)
{
   if (ex.Message.IndexOf("Unable to relay for") > -1)
   {
      this.Cursor = Cursors.Default;
      MessageBox.Show("Der SMTP-Server '" + smtpClient.Host +
        "' kann die E-Mail an die angegebene Adresse nicht " +
        "weiterleiten. Wahrscheinlich ist die Weiterleitung " +
        "von E-Mails für die IP-Adresse des Client in der " +
        "Konfiguration des SMTP-Servers untersagt",
        Application.ProductName, MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
   }
   else
   {
      this.Cursor = Cursors.Default;
      MessageBox.Show(this.GetExceptionMessages(ex), Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
catch (Exception ex)
{
   this.Cursor = Cursors.Default;
   MessageBox.Show(this.GetExceptionMessages(ex), Application.ProductName,
      MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Listing 370: Senden einer Mail über die Klasse SmtpClient

Hinweis

Im Beispielordner finden Sie zu diesem Rezept neben dem einfachen Beispiel auch ein Beispiel, das dem Anwender ermöglicht, die E-Mail-Daten in ein Formular einzugeben. Das Formular finden Sie auch in der Codebook-Klassenbibliothek unter dem Namen `Addison_Wesley.Codebook.Forms.SmtpMailForm`. Dieses Formular können Sie natürlich auch gerne kopieren und in Ihren Anwendungen einsetzen.

212 E-Mails über MAPI bzw. Outlook versenden

Viele Unternehmen bieten ihren Mitarbeitern E-Mail-Funktionalitäten nicht über einen SMTP-, POP3- und/oder IMAP-Server an, sondern betreiben einen Exchange-Server. Mails können Sie bei solchen Systemen nur über spezielle Microsoft-Komponenten versenden. Neben der COM-Komponente CDO (Collaborating Data Objects) können Sie dazu MAPI (Messaging API) verwenden. MAPI ist ein wesentlicher Teil des Windows-API zum Zugriff auf die Microsoft-Mail-Features (über Outlook und den Exchange Server). Leider ist die Programmierung sehr aufwändig. Ein von Thomas Scheidegger entwickeltes, hervorragendes Beispiel inklusive einer C#-Klasse zur Vereinfachung des MAPI-Zugriffs finden Sie auf der Website von *The Code Project* an der Adresse *www.codeproject.com/KB/IP/simplemapidotnet.aspx*.

Ich verwende in diesem Rezept die Klasse `Mapi` von Thomas Scheidegger, die als Public Domain veröffentlicht ist, und zeige lediglich, wie Sie eine einfache Mail über die dort enthaltenen Klassen verwenden. Schauen Sie sich auf jeden Fall auch das Beispiel an, das dem Quellcode beiliegt.

Kopieren Sie die Datei *MapiApi.cs* aus einem der beiden Beispiele (der Konsolen- oder der Windows-Anwendung) in den Projektordner Ihres Projekts. Um eine Mail zu versenden (was nur eine der Anwendungsmöglichkeiten dieser Klasse ist), erzeugen Sie eine Instanz der Klasse `Mapi`. Über die Methode `Logon` können Sie das Programm in MAPI einloggen (was aber eigentlich nur dann notwendig ist, wenn der Zugang geschützt ist). Wenn Sie einen spezifischen Benutzer einloggen wollen, übergeben Sie den Handle eines User-Token (als `IntPtr`). Dieses Handle können Sie über die API-Funktion `LogonUser` erzeugen. Informationen dazu finden Sie im Rezept 307. `LogonUser` gibt, wie viele der Methoden der `Mapi`-Klasse, `false` zurück, wenn ein Fehler aufgetreten ist. Diesen (leider in vielen Fällen nicht sehr aussagekräftigen) Fehler können Sie über die Methode `Error` auslesen.

Nun können Sie über die Methode `AddRecip` Empfänger hinzufügen. Dazu übergeben Sie am ersten Argument die Adresse des Empfängers. Unklar dabei ist, dass der Name des ersten Arguments (`name`) eigentlich darauf hinweist, dass hier der Name des Empfängers übergeben wird. Das zweite Argument ist mit `addr` benannt, was suggeriert, dass hier die Adresse übergeben wird. Beim Test auf meinem System funktionierte aber keine andere Variante als die, die ich hier beschreibe. Das Problem wird wohl aber eher an MAPI liegen als an der .NET-Implementierung von Thomas Scheidegger ☺.

Am dritten Argument können Sie mit `true` bestimmen, dass der Empfänger als CC hinzugefügt wird. Übergeben Sie `false`, wird der Empfänger normal hinzugefügt. Ein BCC ist scheinbar nicht möglich.

Über die `Attach`-Methode können Sie Dateien anfügen, indem Sie einfach den Dateinamen übergeben.

Die Nachricht können Sie schließlich über die `Send`-Methode senden. Am ersten Argument übergeben Sie den Betreff, am zweiten den Text der Nachricht. `Send` gibt `false` zurück, wenn beim Senden ein Fehler aufgetreten ist. Den Fehler können Sie wieder über die Methode `Error` auslesen.

Schließlich loggen Sie den eingeloggten Benutzer über `Logoff` wieder aus.

Das Beispielprogramm in Listing 371 benötigt den Import der Namensräume `System`, `System.IO`, `System.Reflection` und `Win32Mapi` und das Einbinden der Datei *MapiApi.cs*.

```
// Mapi-Instanz erzeugen
Mapi mapi = new Mapi();

// Logon in MAPI. Sie können den aktuellen
// Benutzer einloggen, indem Sie IntPtr.Zero übergeben.
// Wollen Sie einen spezifischen Benutzer einloggen,
// können Sie ein Token über die API-Funktion LogonUser
// erzeugen und dessen Handle übergeben.
if(mapi.Logon(IntPtr.Zero) == false)
{
   Console.WriteLine("Der Login in MAPI ist fehlgeschlagen: " +
      mapi.Error());
   return;
}

// Empfänger hinzufügen
string mailAddress = "juergen.bayer@t-online.de";
mapi.AddRecip(mailAddress, null, false);

// Datei anfügen
string fileName = Path.Combine(Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location), "dontpanic.gif");
mapi.Attach(fileName);

// Mail senden
string subject = "Party";
string message = "Hallo Zaphod, hast du Lust auf eine Party im Restaurant " +
   "am Ende der Galaxis?";
if (mapi.Send(subject, message))
{
   Console.WriteLine("E-Mail erfolgreich versendet");
}
else
{
```

Listing 371: Senden einer Mail über MAPI und die Klasse Mapi von Thomas Scheidegger

```
    Console.WriteLine("Das Senden ist fehlgeschlagen: {0}", mapi.Error());
}
// Aus MAPI ausloggen
mapi.Logoff();
```

Listing 371: Senden einer Mail über MAPI und die Klasse Mapi von Thomas Scheidegger (Forts.)

Das Senden der E-Mail erfolgt dann normalerweise über Outlook. Dabei erscheint u. U. ein Sicherheitshinweis (Abbildung 120).

Dieser Sicherheitshinweis, bei dem der Ja-Schalter für fünf Sekunden deaktiviert ist, ist ab Outlook 2000, Service Pack 2 enthalten. Microsoft will damit verhindern, dass Virenprogramme unbemerkt E-Mails versenden können. Sie können den Dialog leider nicht unterbinden.

Sie können jedoch an der Adresse *www.express-soft.com/mailmate/clickyes.html* das Tool *Express ClickYes* downloaden, das die Betätigung des Ja-Schalters automatisch vornimmt. Nach der Installation finden Sie dieses Tool im Infobereich der Taskleiste. Wählen Sie im Kontextmenü dieses Eintrags die Option START SUSPENDED ab und schalten Sie die Option START ON LOGON ein, damit *Express ClickYes* beim nächsten Windows-Start automatisch ausgeführt wird. Nach der Installation müssen Sie einmal den Eintrag RESUME im Kontextmenü anklicken, damit *Express ClickYes* ohne einen Neustart arbeitet.

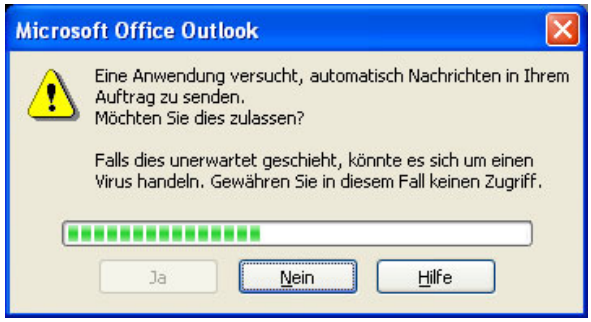


Abbildung 120: Der Sicherheitshinweis von Outlook ab Version 2000, SP 2

213 Browser starten

Den Default-Browser des Systems können Sie sehr einfach über die statische `Start`-Methode der `Process`-Klasse (aus dem Namensraum `System.Diagnostics`) starten. Als »Dateiname« geben Sie am ersten Argument die komplette Webadresse an (inklusive http://, ftp://, file:// etc.). Wenn Sie einen spezifischen Browser starten wollen, erzeugen Sie ein neues `Process`-Objekt, definieren in dessen `StartInfo`-Eigenschaft in der Eigenschaft `FileName` den Dateinamen und geben in der Eigenschaft `Arguments` die Webadresse an.

```
// Standard-Browser öffnen
Process.Start("http://www.google.de");

// Internet Explorer öffnen
Process process = new Process ();
process.StartInfo.FileName = "iexplore.exe";
process.StartInfo.Arguments = "http://www.addison-wesley.de";
process.Start();
```

Listing 372: Starten des Standard- und eines speziellen Browsers

Eine andere Möglichkeit, die ich hier nicht beschreibe, ist übrigens das Fernsteuern des Internet Explorers ähnlich dem Fernsteuern von Word und Excel, das ich im Kapitel »COM-Interop mit Office« beschreibe. Das Ganze ist aber nicht allzu kompliziert, da Sie beim Internet Explorer auf dieselben Methoden und Eigenschaften zugreifen wie beim Webbrowser-Steuerelement, das ich im folgenden Rezept beschreibe.

214 Webseiten (HTML-Dokumente) in der Anwendung darstellen

Über das `WebBrowser`-Steuerelement können Sie HTML-Dokumente aus einer lokalen (Datei-) oder einer Internet-Quelle in einer Anwendung darstellen. Dieses Steuerelement verwendet dasselbe ActiveX-Steuerelement, das auch der Internet Explorer einsetzt, und bietet deswegen eine große Anzahl an Möglichkeiten (wie zum Beispiel auch das Ausdrucken und Speichern von HTML-Dokumenten).

Tabelle 25 beschreibt die wichtigsten Eigenschaften, Tabelle 26 die wichtigsten Methoden und Tabelle 27 die wichtigsten Ereignisse dieses Steuerelements.

Eigenschaft	Beschreibung
`AllowNavigation`	bestimmt, ob das Steuerelement zu einer weiteren Seite navigieren kann nach dem die Initial-Seite geladen wurde. Die Voreinstellung ist `true`.
`AllowWebBrowser-Drop`	bestimmt, ob das Steuerelement zu Seiten navigieren kann, die auf ihm (per Drag & Drop z. B. aus der Adressleiste des Internet Explorers) fallen gelassen werden. Die Voreinstellung ist `true`.
`CanGoBack`	Über diese Eigenschaft können Sie feststellen, ob es im aktuellen Zustand möglich ist, über die `GoBack`-Methode zu einer vorherigen Seite zurückzugehen.
`CanGoForward`	Ähnlich `CanGoBack` können Sie über diese Eigenschaft feststellen, ob es gerade möglich ist, über die `GoForward`-Methode zur nächsten Seite zu gehen.
`Document`	liefert ein `HtmlDocument`-Objekt, das das aktuell angezeigte HTML-Dokument repräsentiert.
`IsBusy`	gibt an, ob der `WebBrowser` gerade mit dem Herunterladen eines HTML-Dokuments oder einer Datei beschäftigt ist.
`ReadyState`	liefert den aktuellen Status des Steuerelements.

Tabelle 25: Die wichtigsten Eigenschaften des WebBrowser-Steuerelements

Eigenschaft	Beschreibung
ScriptErrorsSuppressed	bestimmt, ob Script-Fehler, die im Dokument auftreten, unterdrückt werden. Die Voreinstellung ist false.
WebBrowserShortcutsEnabled	gibt an, ob (die vom Internet Explorer bekannten) Tastatur-Shortcuts verfügbar sind. Die Voreinstellung ist true.

Tabelle 25: Die wichtigsten Eigenschaften des WebBrowser-Steuerelements (Forts.)

Methode	Beschreibung
GoBack()	navigiert zu der zuvor angezeigten Seite.
GoForward()	navigiert zu der nächsten Seite in der Seiten-Reihenfolge, sofern zuvor zu einer vorherigen Seite navigiert wurde.
GoHome()	navigiert zu dem im Internet Explorer eingestellten Home-Dokument.
GoSearch()	navigiert zu der im Internet Explorer eingestellten Standard-Such-Seite.
Navigate (string *urlString*) Navigate (string *urlString*, bool *newWindow*)	Über diese Methode können Sie das Steuerelement zu einer Seite navigieren. In der hier dargestellten zweiten Variante können Sie bestimmen, dass die Seite in einer neuen Instanz des Internet Explorers dargestellt wird. Darüber hinaus existieren noch weitere Varianten der Navigate-Methode, denen Sie zum Beispiel eine Uri-Instanz anstelle des URL-String übergeben.
Refresh()	lädt das gerade angezeigte Dokument neu.
ShowPrintDialog()	zeigt den Drucken-Dialog des Internet Explorers an und führt zum Ausdruck des Dokuments, sofern der Benutzer den Drucken-Schalter betätigt.
ShowPrintPreviewDialog()	zeigt den Druck-Vorschaudialog des Internet Explorers an.
ShowSaveAsDialog()	zeigt den Speichern-Unter-Dialog des Internet Explorers an und führt zum Speichern des Dokuments, sofern der Benutzer den Speichern-Schalter betätigt.
Stop()	bricht eine aktuell laufende Aktion, wie zum Beispiel das Laden eines Dokuments, ab.

Tabelle 26: Die wichtigsten Methoden des WebBrowser-Steuerelements

Ereignis	Beschreibung
CanGoBackChanged	wird aufgerufen, wenn die Verfügbarkeit des Zurück-Befehls (GoBack) sich ändert. Über die Eigenschaft CanGoBack können Sie abfragen, ob es möglich ist, zu der vorherigen Seite zurückzugehen.
CanGoForwardChanged	wird aufgerufen, wenn die Verfügbarkeit des Vor-Befehls (GoForward) sich ändert. Über die Eigenschaft CanGoForward können Sie abfragen, ob es möglich ist, zu der nächsten Seite zu gehen.
DocumentCompleted	wird aufgerufen, wenn das Laden eines Dokuments beendet ist.
Navigated	wird aufgerufen, wenn das Steuerelement zu einer neuen Seite navigiert hat und beginnt diese zu laden,

Tabelle 27: Die wichtigsten Ereignisse des WebBrowser-Steuerelements

Ereignis	Beschreibung
`Navigating`	wird aufgerufen bevor das Steuerelement zu einer neuen Seite navigiert
`NewWindow`	wird aufgerufen, bevor das Steuerelement eine Seite in einem neuen (Internet-Explorer-)Fenster lädt.
`TitleChanged`	wird aufgerufen, nachdem der Titel der aktuell angezeigten Seite geändert wurde (also in der Regel, nachdem eine Seite im Steuerelement angezeigt wurde).

Tabelle 27: Die wichtigsten Ereignisse des WebBrowser-Steuerelements (Forts.)

Achtung

Eigenartigerweise erscheinen die Ereignisse `CanGoBackChanged` und `CanGoForwardChanged` in der Version 8.0.50727.42, dem ersten Release von Visual Studio 2008, nicht in der Liste der Ereignisse im Eigenschaften-Fenster. Trotzdem können Sie diese Ereignisse verwenden, indem Sie die Zuweisung im Programmcode von Hand vornehmen.

Die Listings ab Listing 373 zeigen die wichtigsten Features des Steuerelements. Das Beispiel ist in einer Windows-Anwendung implementiert, die in Abbildung 121 dargestellt wird.



Abbildung 121: Das Webbrowser-Beispielprogramm

Über die `Navigate`-Methode, die in einigen Überladungen vorkommt, können Sie zu einer Website navigieren. Einer Variante dieser Methode übergeben Sie die URL als String am ersten Argument. Die Navigation übernehmen im Beispiel das `Click`-Ereignis des Öffnen-Schalters und das `KeyPress`-Ereignis der `TextBox` (*urlTextBox*).

Das Programm benötigt die für Windows.Forms-Anwendungen notwendigen Referenzen und Importe.

```
private void openButton_Click(object sender, System.EventArgs e)
{
   // Navigieren
   this.webBrowser.Navigate(urlTextBox.Text);
}

private void urlTextBox_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
   if (e.KeyChar == '\r')
   {
      e.Handled = true; // Warnton verhindern

      // Navigieren
      this.webBrowser.Navigate(this.urlTextBox.Text);
   }
}
```

Listing 373: Navigieren zu einer Webseite

Die Schalter zum Wechseln zur Startseite (*Home*), zur vorherigen Seite (*Zurück*) und zur nächsten Seite (*Vor*) und zum Stoppen des aktuellen Vorgangs rufen die Methoden `GoHome`, `GoBack`, `GoForward` bzw. `Stop` auf:

```
private void homeButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.GoHome();
}

private void backButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.GoBack();
}

private void forwardButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.GoForward();
}

private void stopButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.Stop();
}
```

Listing 374: Wechseln zur Startseite, zur vorherigen und zur nächsten Seite

Das erneute Laden einer Webseite können Sie über die Refresh-Methode erreichen:

```
private void reloadButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.Refresh();
}
```

Listing 375: Erneutes Laden der aktuellen Webseite

Im Beispiel wird die aktuelle Seite über den Aufruf der Print-Methode direkt ausgedruckt. Sie können aber auch die ShowPrintDialog- oder die ShowPrintPreviewDialog-Methode aufrufen, wenn Sie einen Dialog zur Auswahl des Druckers oder eine Druckvorschau anzeigen wollen. Das Speichern des aktuellen Dokuments ist über den Aufruf der ShowSaveAsDialog-Methode realisiert.


```
private void printButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.Print();
}

private void saveButton_Click(object sender, System.EventArgs e)
{
   this.webBrowser.ShowSaveAsDialog();
}
```

Listing 376: Drucken und Speichern der aktuellen HTML-Seite

Über verschiedene Ereignisse können Sie auf Änderungen des Status reagieren. Das Ereignis Navigating wird z. B. aufgerufen, bevor der WebBrowser zu einer neuen Seite navigiert. Ich verwende dieses Ereignis um den Stopp-Schalter zu aktivieren und den Speichern-, den Reload- und den Drucken-Schalter zu deaktivieren:

```
private void webBrowser_Navigating(object sender,
   WebBrowserNavigatingEventArgs e)
{
   this.stopButton.Enabled = true;
   this.saveButton.Enabled = false;
   this.reloadButton.Enabled = false;
   this.printButton.Enabled = false;
}
```

Listing 377: Aktivieren/Deaktivieren einiger Schalter, bevor eine Seite geladen wird

Das Ereignis DocumentCompleted, das aufgerufen wird, nachdem eine Seite erfolgreich geladen wurde, wird vom Beispiel verwendet, um den Titel des aktuellen Dokuments auszulesen und

die Schalter, die in `Navigating` aktiviert bzw. deaktiviert wurden, zu deaktivieren bzw. zu aktivieren:

```
private void webBrowser_DocumentCompleted(object sender,
   WebBrowserDocumentCompletedEventArgs e)
{
   // Titel des aktuellen Dokuments auslesen
   this.Text = this.webBrowser.Document.Title;

   // Die URL des aktuellen Dokuments in die URL-TextBox schreiben
   this.urlTextBox.Text = this.webBrowser.Url.AbsoluteUri;

   // Stopp-Schalter deaktivieren und den Speichern-,
   // den Reload- und den Drucken-Schalter aktivieren
   this.stopButton.Enabled = false;
   this.saveButton.Enabled = true;
   this.reloadButton.Enabled = true;
   this.printButton.Enabled = true;
}
```

Listing 378: Auslesen des Titels des aktuellen HTML-Dokuments und Aktivieren/Deaktivieren einiger Schalter, nachdem eine Seite geladen wurde

Über die Ereignisse `CanGoForwardChanged` und `CanGoBackChanged` können Sie herausfinden, ob die Verfügbarkeit des Zurück- bzw. Vor-Befehls geändert wurde. Mein Beispiel nutzt diese Ereignisse um die dazu verwendeten Schalter entsprechend zu aktivieren bzw. zu deaktivieren:

```
private void webBrowser_CanGoBackChanged(object sender, System.EventArgs e)
{
   this.backButton.Enabled = this.webBrowser.CanGoBack;
}

private void webBrowser_CanGoForwardChanged(object sender,
   System.EventArgs e)
{
   this.forwardButton.Enabled = this.webBrowser.CanGoForward;
}
```

Listing 379: Aktivieren bzw. Deaktivieren des Zurück- und des Vor-Schalters je nach Verfügbarkeit des entsprechenden Befehls

Und damit ist der einfache Webbrowser fertig. Das Programm lässt sich aber natürlich über die vielfältigen Fähigkeiten des Webbrowser-Steuerelements noch erweitern.

215 Dateien von einem Web- oder FTP-Server über eine WebClient-Instanz downloaden

Die `WebClient`-Klasse (aus dem Namensraum `System.Net`) ermöglicht den einfachen Download (und Upload) von Daten. `WebClient` vereinfacht den Zugriff auf einen Web- oder FTP-Server, der ansonsten über eine Instanz der Klasse `HttpWebRequest` bzw. `FtpWebRequest` auch selbst programmiert werden kann. Diese Klassen werden natürlich von `WebClient` intern eingesetzt. Der Download über eine Instanz der `WebClient`-Klasse ist recht einfach, wobei Sie diesen sogar asynchron ausführen können.

Hinweis

Die Verwendung der `WebClient`-Klasse hat beim Download von einem Webserver einen maßgeblichen Nachteil: Beim (HTTP-)Download von einem Webserver kann es vorkommen, dass der Server oder ein zwischen Server und Client geschalteter Router den Download großer Dateien, besonders bei langsamen Internetverbindungen, einfach unterbricht. Die Methoden zum Download von Daten liefern in diesem Fall eine `WebException` mit der Meldung »Von der Übertragungsverbindung können keine Daten gelesen werden: Eine vorhandene Verbindung wurde vom Remotehost geschlossen.« (in der inneren `Exception`). Sie besitzen keine Möglichkeit, den Download mit einer neuen HTTP-Verbindung ab der Position weiter auszuführen, bis zu der der vorherige Download ausgeführt wurde, obwohl das HTTP-Protokoll den teilweisen Download einer Datei unterstützt. Besonders für Benutzer mit langsamen Internetverbindungen kann das zu einem Ärgernis werden. In der Praxis ist dies in einem Projekt, in dem ich mit entwickelt habe, auch bereits häufiger aufgetreten. Deswegen habe ich den HTTP-Download in einer eigenen Klasse so programmiert, dass dieser mit einer neuen Verbindung immer wieder ab der letzten Position ausgeführt wird, wenn er unterbrochen wurde. Diese Lösung (die eigentlich von Microsoft hätte implementiert werden müssen) finden Sie im Rezept 216.

Hinweis

Aber auch der Download von einem FTP-Server über eine `WebClient`-Instanz führt zu einem Nachteil: Bei einem solchen war in meinen Versuchen das Argument `TotalBytesToReceive` im `DownloadProgressChanged`-Ereignis immer auf -1 gesetzt. Das Argument `ProgressPercentage` speicherte folglich immer 0. Eine vernünftige Fortschrittsmeldung war somit nicht möglich. Das Problem kann daran liegen, dass die `WebClient`-Klasse einen Download im passiven Modus vornimmt (der in der intern eingesetzten `FtpWebRequest`-Instanz Default ist). In diesem Modus werden meist keine Größeninformationen übertragen. Sie könnten den Download über eine `FtpWebRequest`-Instanz selbst programmieren und diesen im aktiven Modus vornehmen (indem Sie die `UsePassive`-Eigenschaft auf `false` stellen). Dabei besteht aber die Gefahr, dass die Kommunikation zwischen dem Client und dem FTP-Server durch Firewalls verhindert wird. Eine einfache Lösung ist das explizite Auslesen der Dateigröße vor dem Download, wie ich es in Rezept 218 zeige.

Hinweis

Eine andere Lösung für Downloads über den *Background Intelligent Transfer Service* (BITS) finden Sie in Rezept 217.

Zum Downloaden von Daten über die `WebClient`-Klasse können Sie die in Tabelle 28 beschriebenen Methoden verwenden.

Methode	Bedeutung
`byte[] DownloadData (string address)` `byte[] DownloadData (Uri address)`	Lädt Daten in ein Byte-Array. Diese Methode wird synchron ausgeführt. Das erhaltene Byte-Array können Sie direkt weiterverarbeiten, zum Beispiel, indem Sie dieses in einen Stream schreiben.
`void DownloadDataAsync (Uri address [, object userToken])`	Lädt Daten asynchron in ein Byte-Array. Während des Downloads wird das `DownloadProgressChanged`-Ereignis aufgerufen. Am Ende des Downloads ruft `WebClient` das `DownloadDataCompleted`-Ereignis auf. Die Eigenschaft `Result` des Ereignisargument-Objekts verwaltet das Byte-Array mit den heruntergeladenen Daten.
`void DownloadFile (string address, string fileName)` `void DownloadFile (Uri address, string fileName)`	Lädt Daten direkt in eine Datei. Diese Methode wird synchron ausgeführt.
`void DownloadFileAsync (Uri address, string fileName [, object userToken])`	Lädt Daten asynchron in eine Datei. Während des Downloads wird das `DownloadProgressChanged`-Ereignis aufgerufen. Am Ende des Downloads ruft `WebClient` das `DownloadFileCompleted`-Ereignis auf.
`string DownloadString (string address)` `string DownloadString (Uri address)`	Lädt Daten synchron in einen String.
`void DownloadStringAsync (Uri address [, object userToken])`	Lädt Daten synchron in einen String. Während des Downloads wird das `DownloadProgressChanged`-Ereignis aufgerufen. Am Ende des Downloads ruft `WebClient` das `DownloadStringCompleted`-Ereignis auf. Die Eigenschaft `Result` des Ereignisargument-Objekts verwaltet den heruntergeladenen String.

Tabelle 28: Die Methoden der WebClient-Klasse zum Download von Daten

Die `DownloadFile`-Methode ermöglicht den einfachen Download einer Datei und das direkte Speichern. Dummerweise überschreibt diese Methode vorhandene Dateien ohne Warnung und erzeugt für den Fall, dass die Datei nicht geschrieben werden kann (weil eine zu überschreibende Datei z. B. schreibgeschützt ist), keine spezielle `IOException`, sondern eine Ausnahme vom Typ `WebException` mit der Fehlermeldung »Während einer WebClient-Anforderung ist ein Ausnahmefehler aufgetreten«. Die eigentliche Fehlermeldung finden Sie aber in der inneren Exception. Die `DownloadData`-Methode ermöglicht hingegen den Download von Daten (oder einer Datei) in Form eines Byte-Arrays, das Sie dann z. B. in einen `FileStream` schreiben oder anderweitig weiter-

verarbeiten können. Beide Methoden (und die zusätzlich verfügbare `DownloadString`-Methode) liegen auch in einer asynchronen Variante vor (`DownloadFileAsync`, `DownloadDataAsync` und `DownloadStringAsync`). Diese Methoden ermöglichen sogar eine Fortschrittsmeldung über das `DownloadProgressChanged`-Ereignis.

Für den einfachen synchronen Download einer Datei erzeugen Sie zunächst eine `WebClient`-Instanz. Falls der Zugriff auf das Internet über einen Proxy-Server erfolgt, erzeugen Sie eine Instanz der `WebProxy`-Klasse, geben die Proxy-Adresse und den Port an und schreiben die Benutzer-Informationen in die Eigenschaft `Credentials` des `WebProxy`-Objekts, die in unserem Fall ein `NetworkCredential`-Objekt aufnimmt.

Erfordert der Zugriff auf die entfernte Ressource eine Authentifizierung (was ja häufig bei FTP-Servern vorkommt), erzeugen Sie ein weiteres `NetworkCredential`-Objekt, initialisieren dies mit den Zugangsdaten und schreiben das Objekt in die Eigenschaft `Credentials`. Für einen anonymen FTP-Zugriff (der über den Benutzer »anonymous« und eine E-Mail-Adresse als Passwort erfolgt) müssen Sie aber keine Benutzerdaten angeben: Beim Zugriff auf einen FTP-Server übergibt die `WebClient`-Klasse automatisch Benutzerdaten für einen anonymen Zugriff, wenn keine speziellen Benutzerdaten angegeben sind.

Den Download der Datei starten Sie schließlich über die `DownloadFile`-Methode, der Sie den URI der entfernten Daten und den Namen der zu erzeugenden Datei übergeben. Da beim Download und beim Speichern der Datei einige Fehler auftreten können, sollten Sie diese natürlich behandeln. Dabei müssen Sie beachten, dass die Meldung der äußeren Exception häufig nicht besonders aussagekräftig ist und die inneren Exceptions die eigentliche Ursache melden. Deswegen verwendet die Beispiel-Methode `DownloadFile` in Listing 380 beim Auftreten einer Exception die Methode `GetExceptionMessages` zum Auslesen der Nachrichten aller Exceptions.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Net` und `System.Windows.Forms` importieren.

```
private void DownloadFile()
{
   // WebClient erzeugen
   WebClient webClient = new WebClient();

   // Die Proxy-Daten übergeben (falls notwendig)
   string proxyAddress = null;
   int proxyPort = 8080;
   string proxyUserName = null;
   string proxyPassword = null;
   string proxyUserDomain = null;
   if (proxyAddress != null && proxyAddress.Length > 0)
   {
      WebProxy webProxy = new WebProxy(proxyAddress, proxyPort);
      webProxy.Credentials = new NetworkCredential(
```

Listing 380: Synchroner Download einer Datei über eine WebClient-Instanz

```
            proxyUserName, proxyPassword,
            proxyUserDomain);
            webClient.Proxy = webProxy;
      }

      // Die Authentifizierungsdaten übergeben (falls notwendig)
      string username = null;
      string password = null;
      if (username != null && username.Length > 0)
      {
         webClient.Credentials = new NetworkCredential(
            username, password);
      }

      try
      {
         // Die Datei herunterladen
         string uri = "http://www.juergen-bayer.net/artikel/CSS/CSS.pdf";
         string destinationFilename = "C:\\Css.pdf";
         webClient.DownloadFile(uri, destinationFilename);

         // Ergebnis melden
         MessageBox.Show("Fertig", Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Information);
      }
      catch (WebException ex)
      {
         // Fehler melden
         MessageBox.Show("Fehler beim Download: " +
            this.GetExceptionMessages(ex),
            Application.ProductName, MessageBoxButtons.OK,
            MessageBoxIcon.Error);
      }
      catch (Exception ex)
      {
         // Fehler melden
         MessageBox.Show(GetExceptionMessages(ex),
            Application.ProductName, MessageBoxButtons.OK,
            MessageBoxIcon.Error);
      }
   }

   private static string GetExceptionMessages(Exception ex)
   {
      string messages = ex.Message;
```

Listing 380: Synchroner Download einer Datei über eine WebClient-Instanz (Forts.)

```
   if (ex.InnerException != null)
   {
      messages += Environment.NewLine;
      messages += GetExceptionMessages(ex.InnerException);
   }
   return messages;
}
```

Listing 380: Synchroner Download einer Datei über eine WebClient-Instanz (Forts.)

Die `WebClient`-Klasse implementiert das ereignisbasierte asynchrone Entwurfsmuster (siehe Rezept 368). Aus diesem Grunde ist der asynchrone Download von Dateien oder Daten sehr einfach. Dazu rufen Sie die entsprechende asynchrone Methode auf (z. B. `DownloadFileAsync`). Diese Methode kehrt relativ schnell zurück und das Programm kann weiter ausgeführt werden. Der Download wird dann im Hintergrund ausgeführt.

Während des Downloads erhalten Sie in mehr oder weniger regelmäßigen Abständen eine Information über den Fortschritt in dem Ereignis `DownloadProgressChanged`. Das Beenden der Methode wird über die Ereignisse `DownloadDataCompleted`, `DownloadFileCompleted` oder `DownloadStringCompleted` signalisiert, je nachdem, welche Methode Sie zum Download aufgerufen haben. Außerdem können Sie den Download über die `CancelAsync`-Methode abbrechen. Aus diesem Grunde sollten Sie die `WebClient`-Instanz in einem privaten Feld verwalten und in Konstruktor der Klasse erzeugen. Dort müssen Sie natürlich die Ereignisse, die Sie verwenden wollen, mit entsprechenden Methoden initialisieren.

Der folgende Quellcode basiert auf einem Formular mit Namen `StartForm`, das mit der `ProgressBar`-Instanz `progressBar` und dem `StatusLabel toolStripStatusLabel` (auf einer `StatusStrip`-Instanz) ausgestattet ist. Zum Kompilieren müssen Sie die Namensräume `System`, `System.Windows.Forms`, `System.IO`, `System.Net` und `System.ComponentModel` importieren und die Methode `GetExceptionMessages` aus dem vorherigen Beispiel integrieren.

```
/* Referenz auf den WebClient */
private WebClient webClient;

public StartForm()
{
   this.InitializeComponent();

   // WebClient erzeugen
   this.webClient = new WebClient();

   // Ereignisse zuweisen
   webClient.DownloadProgressChanged +=
      new DownloadProgressChangedEventHandler(
      this.webClient_DownloadProgressChangedChanged);
```

Listing 381: WebClient-Referenz und Konstruktor eines Formulars, der die WebClient-Instanz erzeugt und die Ereignisse zuweist

```
   webClient.DownloadFileCompleted +=
      new AsyncCompletedEventHandler(
      this.webClient_DownloadFileCompleted);
}
```

Listing 381: WebClient-Referenz und Konstruktor eines Formulars, der die WebClient-Instanz erzeugt und die Ereignisse zuweist (Forts.)

Den direkten Download einer Datei können Sie dann über die `DownloadFileAsync`-Methode starten. Zuvor können Sie natürlich wieder die gegebenenfalls benötigten Proxy-Informationen übergeben:


```
private void DownloadFileAsync()
{
   // Die Proxy-Daten übergeben (falls notwendig)
   string proxyAddress = null;
   int proxyPort = 8080;
   string proxyUserName = null;
   string proxyPassword = null;
   string proxyUserDomain = null;
   if (proxyAddress != null && proxyAddress.Length > 0)
   {
      WebProxy webProxy = new WebProxy(proxyAddress, proxyPort);
      webProxy.Credentials = new NetworkCredential(
         proxyUserName, proxyPassword,
         proxyUserDomain);
      webClient.Proxy = webProxy;
   }

   // Die Authentifizierungsdaten übergeben (falls notwendig)
   string username = null;
   string password = null;
   if (username != null && username.Length > 0)
   {
      this.webClient.Credentials = new NetworkCredential(
         username, password);
   }

   try
   {
      // Das Herunterladen der Datei starten
      Uri uri = new Uri("http://www.juergen-bayer.net/artikel/CSS/CSS.pdf");
      string destinationFilename = "C:\\Css.pdf";
      this.webClient.DownloadFileAsync(uri, destinationFilename);
   }
```

Listing 382: Starten des asynchronen Downloads einer Datei

```
   catch (WebException ex)
   {
      // Fehler melden
      MessageBox.Show("Fehler beim Download: " +
         GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
   catch (Exception ex)
   {
      // Fehler melden
      MessageBox.Show(GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
}
```

Listing 382: Starten des asynchronen Downloads einer Datei (Forts.)

Einen Abbruch des asynchronen Downloads können Sie über einen Aufruf der Methode `CancelAsync` implementieren:

```
private void Cancel()
{
   this.webClient.CancelAsync();
}
```

In der Methode für das `DownloadProgressChanged`-Ereignis erhalten Sie in der Eigenschaft `BytesReceived` des Ereignisargument-Objekts die Anzahl der bisher empfangenen Bytes. Die Eigenschaft `TotalBytesToReceive` liefert die Gesamtzahl der zu empfangenden Bytes, über `ProgressPercentage` können Sie den prozentualen Anteil des aktuellen Fortschritts auslesen. Damit können Sie beispielsweise eine Statusmeldung (in einem `StatusLabel`) ausgeben und eine `ProgressBar` aktualisieren:

```
private void webClient_DownloadProgressChanged (
   object sender, DownloadProgressChangedEventArgs e)
{
   long bytesReceived = e.BytesReceived;
   long totalBytesToReceive = e.TotalBytesToReceive;
   int progressPercentage = e.ProgressPercentage;
   this.toolStripStatusLabel.Text = bytesReceived +
      " Bytes von " + totalBytesToReceive + " empfangen ...";
   this.progressBar.Value = progressPercentage;
}
```

Listing 383: Reaktion auf das DownloadProgressChanged-Ereignis

Hinweis

Wie bereits gesagt, erhalten Sie beim FTP-Download in der Eigenschaft `TotalBytesToReceive` meist (oder immer) den Wert -1. Die Eigenschaft `ProgressPercentage` liefert leider auch nur den Wert 0. Das Problem liegt daran, dass die `ContentLength`-Eigenschaft des intern verwendeten Response-Stream ebenfalls -1 zurückgibt, wahrscheinlich deswegen, weil FTP-Server das Auslesen der Länge beim Herunterladen im passiven Modus nicht unterstützen. Sie können die Größe einer herunterzuladenden Datei jedoch selbst ermitteln. Wie das geht, zeige ich in Rezept 218.

Die Methode für das `DownloadFileCompleted`-Ereignis wird immer dann aufgerufen, wenn der asynchrone Download aus irgendeinem Grund beendet wurde. Dies ist natürlich der Fall, wenn der Download erfolgreich abgeschlossen wurde. `DownloadFileCompleted` wird aber auch aufgerufen, wenn ein Fehler aufgetreten ist oder der Download über die `CancelAsync`-Methode abgebrochen wurde. Deshalb sollten Sie die Eigenschaften `Error` und `Cancelled` des Ereignisargument-Objekts abfragen um herauszufinden, warum der Download beendet wurde:


```
private void webClient_DownloadFileCompleted(object sender,
   System.ComponentModel.AsyncCompletedEventArgs e)
{
   if (e.Cancelled)
   {
      this.toolStripStatusLabel.Text = "Abgebrochen";
   }
   else if (e.Error != null)
   {
      this.toolStripStatusLabel.Text = "Fehler";
      MessageBox.Show(GetExceptionMessages(e.Error),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
   else
   {
      this.toolStripStatusLabel.Text = "Fertig";
   }
}
```

Listing 384: Reaktion auf das DownloadFileCompleted-Ereignis

Tipp

Die `WebClient`-Klasse ermöglicht auch den Download von Dateien über eine SSL-Verbindung (über eine https-Adresse, siehe bei *de.wikipedia.org/wiki/Transport_Layer_Security*). Das bei einer solchen (geschützten) Verbindung vom Server zu seiner Authentifizierung gesendete Zertifikat muss entweder eines sein, das von einer der bekannten Zertifizierungsstellen stammt oder für das auf dem lokalen Rechner eine Vertrauensstellung eingerichtet ist. Der Versuch, über eine SSL-Verbindung mit einem ungültigen (nicht zur Web-Domäne gehörenden, abgelaufenen etc.) Zertifikat eine Datei herunterzuladen, schlägt mit dem folgenden Fehler fehl: »Die zugrunde liegende Verbindung

wurde geschlossen: Für den geschützten SSL/TLS-Kanal konnte keine Vertrauensstellung hergestellt werden«. Die innere Exception meldet den Fehler »Das Remotezertifikat ist laut Validierungsverfahren ungültig.«. Diese Exception ist absolut gerechtfertigt, da das Zertifikat ein wichtiger Beweis dafür ist, dass es sich beim Webserver tatsächlich um den angesprochenen Server handelt. Der Internet Explorer meldet beim Versuch, von einem solchen Webserver über SSL eine Datei herunterzuladen, dass das Zertifikat ungültig ist, und ermöglicht es, die Datei trotzdem herunterzuladen. Dieses Rezept könnte entsprechend erweitert werden, aber bisher habe ich dafür noch keine Lösung gefunden.

Tipp

Den Zugriff über einen Proxy-Server können Sie sehr gut über den Freeware-Server *Sambar* (*www.sambar.de*) testen. Sambar ist ein Web- und ein Proxy-Server, der sehr einfach zu konfigurieren ist und der auch dann problemlos zum Test (zum Beispiel des authentifizierten Zugriffs) verwendet werden kann, wenn der Internetzugriff über eine direkte LAN- oder DSL-Verbindung erfolgt. Achten Sie in der Konfiguration gegebenenfalls darauf, dass Sie den HTTP-Port auf einen anderen Wert als den voreingestellten Port 80 (der standardmäßig vom IIS verwendet wird) einstellen (SERVER / HTTP / SERVER PORT). Unter SERVER / ÜBERBLICK können Sie einstellen, dass Sambar als HTTP-Proxy arbeitet. Außerdem können Sie hier festlegen, dass der Proxy eine Authentifizierung erzwingt (PROXY AUTHENTICATION). Die Benutzer, die sich gegenüber Sambar anmelden können, verwalten Sie über das »Menü« SICHERHEIT / BENUTZER. Stellen Sie gegebenenfalls noch ein, dass der HTTP-Proxy-Zugriff protokolliert wird, damit Sie diesen überprüfen können (SERVER / PROXIES / LOG HTTP PROXY USAGE).

Hinweis

In Ihren Programmen sollten Sie für den Zugriff über einen Proxy-Server beachten, dass die Klasse `HttpWebRequest` grundsätzlich die im Internet Explorer eingestellten Proxy-Einstellungen verwenden, wenn keine spezifischen Proxy-Einstellungen angegeben werden.

Hinweis

Eine Alternative zur Verwendung der `WebClient`-Klasse für einen FTP-Download ist die `FtpRequest`-Klasse, über die Sie mehr Möglichkeiten zur Arbeit mit FTP-Servern haben. Bei einem Download können Sie die Daten, die Sie in dem Response-Stream eines `FtpResponse`-Objekts erhalten, in einzelnen Blöcken auslesen und verarbeiten. Diese Möglichkeit wird in diesem Buch nicht behandelt.

Hinweis

So richtig funktioniert der FTP-Download mit .NET leider (noch) nicht. Der Versuch, Dateien von dem Server *ftp.fu-berlin.de* herunterzuladen, resultierte z. B. in der Exception »Der Server hat eine Protokollverletzung ausgeführt«. Und das, obwohl ein anderer »FTP-Client« (der Total Commander) ohne Probleme Dateien von diesem Server herunterladen konnte.

216 Dateien von einem Webserver über eine HttpWebRequest-Instanz downloaden

Die `WebClient`-Klasse (aus dem Namensraum `System.Net`) ermöglicht auf eine recht einfache Weise den (synchronen und asynchronen) Download von Dateien unter anderem von einem Webserver (siehe Rezept 215). Die Verwendung dieser Klasse beinhaltet aber ein in der Praxis maßgebliches Problem: Bei Downloads großer Dateien (mit mehreren MB) unterbrechen manche Webserver oder die zwischen Webserver und Client geschalteten Router die HTTP-Verbindung nach einiger Zeit (wobei es mir bisher nicht gelungen ist, herauszufinden, woran das liegt). Die Methoden einer `WebClient`-Instanz werfen dann eine `WebException` mit der Meldung »Von der Übertragungsverbindung können keine Daten gelesen werden: Eine vorhandene Verbindung wurde vom Remotehost geschlossen.« (in der inneren `Exception`). In diesem Fall müssen Sie den Download erneut ausführen. Sie besitzen jedoch keine Möglichkeit den Download ab der Stelle auszuführen, an der der vorherige Versuch abgebrochen ist. Dieses Manko ist nicht zu unterschätzen: Ein Anwender, dessen minutenlanger Download bei 90% abbricht, ist sehr verärgert. Moderne Downloadmanager wie der des Firefox-Browsers reagieren auf den Abbruch der Verbindung einfach dadurch, dass sie eine neue aufbauen und den Download an der Position starten, an der der letzte unterbrochen wurde. Das HTTP-Protokoll unterstützt dies, aber leider nicht die `WebClient`-Klasse.[2]

Um das Problem des HTTP-Verbindungsabbruchs zu lösen gehe in diesem Rezept einen flexibleren Weg. Sie können Daten von einem Webserver auch über ein `HttpWebRequest`-Objekt anfordern. Über die `GetResponse`-Methode eines solchen Objekts erhalten Sie die Antwort des Servers in Form eines `HttpWebResponse`-Objekts. Dessen `GetResponseStream`-Methode liefert einen Stream, aus dem Sie die Daten lesen können. Im Prinzip ist das nichts anderes als das, was die Download-Methoden der `WebClient`-Klasse intern machen.

Das Problem des Unterbrechens der Verbindung beim Download großer Dateien können Sie über die Methode `AddRange` der `HttpWebRequest`-Instanz lösen, der Sie einen Bereich zu lesender Daten oder eine Position übergeben können. Eine positive Zahl definiert eine Position, ab der die Datei heruntergeladen wird. Bei einer Unterbrechung des Downloads (die über eine `IOException` gemeldet wird) können Sie diese Technik nutzen um auch mehrfach unterbrochene Downloads sicher bis zum Ende auszuführen.

Hinweis

Eine andere Möglichkeit, Dateien im Hintergrund mit einer gewissen Eigenintelligenz herunterzuladen, ist die Verwendung der *Background Intelligend Transfer Services* (BITS), die ich in Rezept 217 beschreibe.

Die Klasse `HttpDownload`, die ich in der Praxis entwickelt habe und im Folgenden beschreibe, arbeitet möglichst flexibel. Sie ermöglicht deshalb den synchronen und asynchronen Download einer Datei über das HTTP-Protokoll in einen beliebigen Stream. Außerdem können für den Fall, dass zwischen dem Rechner und dem Internet ein Proxy-Server liegt, die notwendigen Proxy-Informationen übergeben werden. Für den Fall, dass der Download vom Server

2. Dieses Problem ist u. U. über das Setzen eines HTTP-Headers lösbar. Die HTTP-Header erreichen Sie über die `Headers`-Eigenschaft der `WebClient`-Klasse.

unterbrochen wird, ermöglicht die Klasse eine automatische Weiter-Ausführung des Downloads in einem erneuten Versuch. Die Anzahl der Versuche wird beim Aufruf der Methode übergeben.

Um die Anwendung der Klasse für den asynchronen Download möglichst komfortabel zu machen, implementiert `HttpDownload` das ereignisbasierte asynchrone Entwurfsmuster. Damit kann der anwendende Programmierer in den Ereignismethoden normal programmieren, also auf Steuerelemente zugreifen, ohne sich darum kümmern zu müssen, dass diese in dem richtigen Thread angesprochen werden (was bei der Verwendung eines Thread ansonsten der Fall wäre). Dieses Entwurfsmuster macht die Programmierung der Klasse leider etwas unübersichtlich. Grundlagen zu dem ereignisbasierten asynchronen Entwurfsmuster finden Sie im Rezept 368. Ich verzichte deswegen bei der Beschreibung der Klasse `HttpDownload` auf eine nähere Erläuterung dieses Entwurfsmusters.

Zum Kompilieren der `HttpDownload`-Klasse und der von ihr verwendeten Typen müssen Sie die Namensräume `System`, `System.Net`, `System.IO`, `System.ComponentModel` und `System.Threading` importieren.

Die Implementierung beginnt mit einer Aufzählung, die im `DownloadProgressChanged`-Ereignis eine Information darüber gibt, was die `DownloadAsync`-Methode gerade macht:

```
public enum DownloadState
{
   /* Die Verbindung wird zurzeit geöffnet */
   OpeningConnection,


   /* Es werden zurzeit Daten gelesen */
   ReadingData
}
```

Listing 385: Aufzählung für den Fortschritt der Download-Methode

Das `DownloadProgressChanged`-Ereignis erhält einige Informationen über den aktuellen Zustand. Deswegen habe ich die Klasse `DownloadProgressChangedEventArgs` für die Ereignis-Argumente und den Delegaten `DownloadProgressChangedEventHandler` implementiert:

```
/* Ereignisargument-Klasse für das DownloadProgressChangedChanged-Ereignis */
public class DownloadProgressChangedEventArgs: ProgressChangedEventArgs
{
   /* Der URI, von dem die Daten heruntergeladen werden */
   public readonly Uri Uri;

   /* Gibt die Anzahl empfangener Bytes an */
   public readonly long BytesReceived;
```

Listing 386: Ereignisargument-Klasse und Delegat für das DownloadProgressChangedChanged-Ereignis

```
   /* Gibt die Gesamtzahl der Bytes an, die empfangen werden müssen */
   public readonly long TotalBytesToReceive;

   /* Gibt den aktuellen Status des Downloads an */
   public readonly DownloadState DownloadState;

   /* Gibt an, um welchen Versuch es sich handelt */
   public readonly int DownloadAttemptNumber;

   /* Gibt an, wie viele Download-Versuche maximal ausgeführt werden dürfen */
   public readonly int MaxDownloadAttempts;

   /* Konstruktor */
   internal DownloadProgressChangedEventArgs(Uri uri, long bytesReceived,
      long totalBytesToReceive, int progressPercentage,
      DownloadState downloadState, int downloadAttemptNumber,
      int maxDownloadAttempts, object userState)
      : base(progressPercentage, userState)
   {
      this.Uri = uri;
      this.BytesReceived = bytesReceived;
      this.TotalBytesToReceive = totalBytesToReceive;
      this.DownloadState = downloadState;
      this.DownloadAttemptNumber = downloadAttemptNumber;
      this.MaxDownloadAttempts = maxDownloadAttempts;
   }
}

/* Delegat für das DownloadProgressChanged-Ereignis */
public delegate void DownloadProgressChangedEventHandler(object sender,
   DownloadProgressChangedEventArgs e);
```

Listing 386: Ereignisargument-Klasse und Delegat für das DownloadProgressChangedChanged-Ereignis (Forts.)

Dem `DownloadCompleted`-Ereignis werden der URI der angeforderten Datei und ein Stream übergeben. In den Stream schreibt die Download-Methode die geladenen Daten. Außerdem besitzt dieses Ereignis die Argumente, die für das ereignisbasierte asynchrone Entwurfsmuster notwendig sind. Deswegen wird `DownloadCompletedEventArgs` von der Klasse `AsyncCompletedEventArgs` abgeleitet und erbt damit die Eigenschaften `Error` (vom Typ `Exception`), `Cancelled` (vom Typ `bool`) und `UserState` (vom Typ `object`). Über das Argument `Error` wird dem `DownloadCompleted`-Ereignis eine aufgetretene Exception übergeben, `Cancelled` wird auf `true` gesetzt, wenn die asynchrone Ausführung abgebrochen wurde, und `UserState` enthält die benutzerdefinierten Daten, die der anwendende Programmierer beim Aufruf der `DownloadAsync`-Methode übergeben hat.

Neben der Ereignisargument-Klasse habe ich natürlich noch den Delegaten für das Ereignis implementiert:

```
/* Ereignisargument-Klasse für das DownloadCompleted-Ereignis */
public class DownloadCompletedEventArgs: AsyncCompletedEventArgs
{
   /* Der URI, von dem die Daten heruntergeladen werden */
   public readonly Uri Uri;

   /* Referenz auf den Ergebnis-Stream */
   public readonly Stream DestinationStream;

   /* Konstruktor */
   public DownloadCompletedEventArgs(Uri uri, Exception error, bool cancelled,
      Stream destinationStream, object userState)
      : base(error, cancelled, userState)
   {
      this.Uri = uri;
      this.DestinationStream = destinationStream;
   }
}

/* Delegat für das DownloadCompleted-Ereignis */
public delegate void DownloadCompletedEventHandler(
   object sender, DownloadCompletedEventArgs e);
```

Listing 387: Ereignisargument-Klasse und Delegat für das DownloadCompleted-Ereignis

Die Klasse `HttpDownload` beginnt mit der Deklaration einiger Eigenschaften, eines privaten Feldes und der Ereignisse. Ich denke, die Kommentare sprechen für sich. Die Bedeutung erschließt sich für alle Eigenschaften außer `IsBusy` aus der `DownloadInternal`-Methode, die den Download ausführt und auf diese Eigenschaften ausliest. `IsBusy` wird verwendet um den Mehrfach-Aufruf der Arbeitsmethode `DownloadInternal` (ab Listing 391) zu verhindern.

```
public class HttpDownload
{
   /* Verwaltet gegebenenfalls notwendige Proxy-Informationen */
   public IWebProxy Proxy;

   /* Verwaltet gegebenenfalls notwendige Informationen zur
      Authentifizierung gegenüber dem Webserver */
   public ICredentials Credentials;

   private int blockSize = 1048576;
   /* Gibt die Größe der auszulesenden Blöcke in Byte an */
   public int BlockSize
   {
      get { return this.blockSize; }
```

Listing 388: Die Eigenschaften, Felder und Ereignisse der HttpDownload-Klasse

```
      set
      {
         if (value >= 1024)
         {
            this.blockSize = value;
         }
         else
         {
            throw new ArgumentException(
               "BlockSize muss ein Wert größer/gleich 1024 sein");
         }
      }
   }

   private int requestTimeout = 100000;
   /* Gibt den Timeout des Request beim Aufruf der
     GetResponse-Methode in Millisekunden an */
   public int RequestTimeout
   {
      get { return this.requestTimeout; }
      set
      {
         if (value >= 0)
         {
            this.requestTimeout = value;
         }
         else
         {
            throw new ArgumentException(
               "RequestTimeout muss ein Wert größer/gleich 0 sein");
         }
      }
   }

   private int delayBetweenDownloadAttempts = 1000;
   /* Gibt die Pause zwischen einem unterbrochenen und einem
      neuen Downloadversuch in Millisekunden an */
   public int DelayBetweenDownloadAttempts
   {
      get { return this.delayBetweenDownloadAttempts; }
      set
      {
         if (value >= 0)
         {
            this.delayBetweenDownloadAttempts = value;
```

Listing 388: Die Eigenschaften, Felder und Ereignisse der HttpDownload-Klasse (Forts.)

```
            }
            else
            {
               throw new ArgumentException("DelayBetweenDownloadAttempts " +
                  "muss ein Wert größer/gleich 0 sein");
            }
         }
      }

      private bool isBusy;
      /* Gibt an, ob der Download gerade ausgeführt wird */
      public bool IsBusy
      {
         get { return this.isBusy; }
      }

      /* Wird von der CancelAsync gesetzt um die asynchrone Ausführung
         abzubrechen */
      private bool cancellationPending;

      /* Wird bei einem Fortschritt des Downloads aufgerufen */
      public event DownloadProgressChangedEventHandler DownloadProgressChanged;

      /* Wird beim Beenden des Downloads aufgerufen */
      public event DownloadCompletedEventHandler DownloadCompleted;
```

Listing 388: Die Eigenschaften, Felder und Ereignisse der HttpDownload-Klasse (Forts.)

Da der Aufruf der Ereignisse dem Ereignisaufruf-Muster von Microsoft entsprechen soll, habe ich zwei geschützte virtuelle Methoden implementiert, die die Ereignisse aufrufen:

```
      /* Ruft das DownloadProgressChanged-Ereignis auf */
      protected virtual void OnDownloadProgressChanged(
         DownloadProgressChangedEventArgs e)
      {
         // Ereignis aufrufen
         if (this.DownloadProgressChanged != null)
         {
            this.DownloadProgressChanged(this, e);
         }
      }

      /* Ruft das DownloadCompleted-Ereignis auf */
      protected virtual void OnDownloadCompleted(DownloadCompletedEventArgs e)
      {
```

Listing 389: Methoden zum Aufruf der Ereignisse der HttpDownload-Klasse

```
      // Busy-Flag zurücksetzen
      this.isBusy = false;

      // Ereignis aufrufen
      if (this.DownloadCompleted != null)
      {
         this.DownloadCompleted(this, e);
      }
   }
```

Listing 389: Methoden zum Aufruf der Ereignisse der HttpDownload-Klasse (Forts.)

Um die Arbeitsmethode (`DownloadInternal`) asynchron aufrufen zu können benötigt die Klasse einen Delegaten, der die Signatur der Arbeitsmethode besitzt. In unserem Fall werden der Arbeitsmethode die Argumente `uri` (URI der angeforderten Daten), `destinationStream` (Referenz auf einen Stream, in den die Daten geschrieben werden), `maxDownloadAttempts` (maximale Anzahl an Download-Versuchen) und das für den Aufruf der Ereignisse notwendige `AsyncOperation`-Objekt übergeben:

```
private delegate void DownloadInternalCallerHandler(
   Uri uri, Stream destinationStream, byte maxDownloadAttempts,
   AsyncOperation asyncOperation);
```

Innerhalb der Arbeitsmethode werden die Ereignisaufruf-Methoden über die `Post`-Methode des übergebenen `AsyncOperation`-Objekts aufgerufen. Dieser Methode wird eine Referenz auf einen Delegaten vom Typ `SendOrPostCallback` übergeben. Dieser Delegat muss innerhalb der Klasse deklariert werden. Natürlich wird auch jeweils eine Methode benötigt, die der Delegat referenziert. `HttpDownload` implementiert deswegen die Delegaten `downloadProgressChangedCallerHandler` und `downloadCompletedCallerHandler` und die Methoden `DownloadProgressChangedCaller` und `DownloadCompletedCaller` zum Aufruf der Ereignisse. Im Konstruktor werden die Delegaten erzeugt:

```
/* Über diesen Delegaten ruft die Arbeitsmethode (DownloadInternal) das
   DownloadProgressChanged-Ereignis auf */
private SendOrPostCallback downloadProgressChangedCallerHandler;

/* Diese Methode wird über den Delegaten
   downloadProgressChangedCallerHandler aufgerufen */
private void DownloadProgressChangedCaller(object state)
{
   // Das DownloadCompleted-Ereignis über die Ereignisaufruf-
   // Methode aufrufen
   this.OnDownloadProgressChanged(
      (DownloadProgressChangedEventArgs)state);
}
```

Listing 390: Delegaten und Methoden zum threadsicheren Aufruf der Ereignisse und Konstruktor oder Klasse HttpDownload

```
/* Über diesen Delegaten ruft die Arbeitsmethode (DownloadInternal) das
   DownloadCompleted-Ereignis auf */
private SendOrPostCallback downloadCompletedCallerHandler;

/* Diese Methode wird über den Delegaten
   DownloadCompletedCallerHandler aufgerufen */
private void DownloadCompletedCaller(object state)
{
   // Das DownloadCompleted-Ereignis über die Ereignisaufruf-
   // Methode aufrufen
   this.OnDownloadCompleted(
      (DownloadCompletedEventArgs)state);
}

/* Konstruktor. Erzeugt die Delegaten zum Aufruf des
   DownloadProgressChanged- und des DownloadCompleted-Ereignisses */
public HttpDownload()
{
   this.downloadProgressChangedCallerHandler =
      new SendOrPostCallback(this.DownloadProgressChangedCaller);
   this.downloadCompletedCallerHandler =
      new SendOrPostCallback(this.DownloadCompletedCaller);
}
```

Listing 390: Delegaten und Methoden zum threadsicheren Aufruf der Ereignisse und Konstruktor oder Klasse HttpDownload (Forts.)

Nun folgt (endlich) die Methode `DownloadInternal`, die den Download ausführt. Diese Methode setzt zunächst das für den Abbruch verwendete Flag `cancellationPending` zurück (dieses Flag wird von der `CancelAsync`-Methode auf `true` gesetzt) und deklariert einige benötigte Variablen.

In der dann beginnenden äußeren Schleife wird zunächst der Download-Versuchszähler inkrementiert. Dann überprüft die Methode, ob abgebrochen werden soll. Ist dies der Fall, ruft `DownloadInternal` über die `Post`-Methode des übergebenen `AsyncOperation`-Objekts das `DownloadCompleted`-Ereignis auf und übergibt am dritten Argument des Konstruktors des übergebenen `DownloadCompletedEventArgs`-Objekts (`Cancelled`) `true`. Die Ausführung der Methode wird in diesem Fall natürlich unterbrochen.

Soll nicht abgebrochen werden, meldet `DownloadInternal` den Fortschritt über das `DownloadProgressChanged`-Ereignis, natürlich auch wieder über die `Post`-Methode des `AsyncOperation`-Objekts. Das Argument `downloadState` des übergebenen `DownloadProgressChangedEventArgs`-Objekts wird dabei auf `DownloadState.OpeningConnection` gesetzt um zu definieren, dass die Methode dabei ist, die Verbindung zu öffnen.

```
private void DownloadInternal(Uri uri, Stream destinationStream,
   byte maxDownloadAttempts, AsyncOperation asyncOperation)
```

Listing 391: Erster Teil der DownloadInternal-Methode

```
{
   // Abbruch-Flag zurücksetzen
   this.cancellationPending = false;

   HttpWebRequest request = null;
   HttpWebResponse response = null;
   Stream responseStream = null;
   int totalBytesReceived = 0;
   byte downloadAttemptNumber = 0;
   long totalBytesToReceive = 0;
   int progressPercentage = 0;
   try
   {
      while (true)
      {
         // Versuchs-Zähler inkrementieren
         downloadAttemptNumber++;

         // Auf eine Abbruch-Anforderung reagieren
         if (this.cancellationPending)
         {
            if (asyncOperation != null)
            {
               // Das DownloadCompleted-Ereignis aufrufen
               asyncOperation.Post(this.downloadCompletedCallerHandler,
                  new DownloadCompletedEventArgs(uri, null, true,
                  destinationStream, asyncOperation.UserSuppliedState));
            }

            // und raus
            return;
         }

         if (asyncOperation != null)
         {
            // Fortschritt melden
            asyncOperation.Post(this.downloadProgressChangedCallerHandler,
               new DownloadProgressChangedEventArgs(uri,
               totalBytesReceived, totalBytesToReceive,
               progressPercentage,
               DownloadState.OpeningConnection, downloadAttemptNumber,
               maxDownloadAttempts, asyncOperation.UserSuppliedState));
         }
```

Listing 391: Erster Teil der DownloadInternal-Methode (Forts.)

Dann erzeugt `DownloadInternal` das für den Download verwendete `HttpWebRequest`-Objekt, initialisiert den Timeout und übergibt die Authentifizierungs- und Proxy-Daten. Für den Fall,

dass der Download aufgrund eines Abbruchs erneut ausgeführt werden soll, bestimmt DownloadInternal über die AddRange-Methode des HttpWebRequest-Objekts den Bereich der herunterzuladenden Daten. Die Variante der AddRange-Methode, der lediglich ein Argument übergeben wird, erwartet die Startposition der Daten. An diesem Argument kann die Anzahl der bisher gelesenen Bytes übergeben werden, da die Startposition (wie in C# üblich) ab 0 gezählt wird.

```
// HttpWebRequest-Instanz für den Download erzeugen
request = (HttpWebRequest)WebRequest.Create(uri);
request.Timeout = this.requestTimeout;

// Proxy-Informationen übergeben
request.Proxy = this.Proxy;

// Die Zugangsdaten für eine gegebenenfalls notwendige
// Authentifizierung übergeben
request.Credentials = this.Credentials;

// Für den Fall, dass der Download aufgrund
// eines Abbruchs mehrfach ausgeführt wird,
// wird hier die Position übergeben, ab
// der heruntergeladen werden soll
request.AddRange(totalBytesReceived);
```

Listing 392: Zweiter Teil der DownloadInternal-Methode

Nachdem DownloadInternal wieder auf eine Abbruch-Anforderung reagiert hat, fordert sie die Antwort (den Response) über die GetResponse-Methode des HttpWebRequest-Objekts an. Da dabei die Verbindung zu dem Webserver aufgebaut wird, habe ich die Anforderung in einem try-Block implementiert. Für den Fall, dass eine WebException auftritt, ist es wahrscheinlich, dass der Server nicht geantwortet hat. In diesem Fall wird die äußere Schleife einfach ab der nächsten Iteration weiter ausgeführt, sofern die Anzahl der Download-Versuche die maximale Anzahl noch nicht erreicht hat. Ist diese Anzahl allerdings erreicht, erzeugt DownloadInternal eine WebException mit einer passenden Fehlermeldung. Wurde beim Aufruf ein AsyncOperation-Objekt übergeben (beim synchronen Aufruf wird dieses Objekt nicht übergeben), ruft die Methode das DownloadCompleted-Ereignis auf und übergibt die WebException am zweiten Argument des erzeugten DownloadCompletedEventArgs-Objekts. Im anderen Fall (für den synchronen Aufruf) wirft DownloadInternal die WebException und setzt das Busy-Flag zurück.

```
// Auf eine Abbruch-Anforderung reagieren
if (asyncOperation != null && this.cancellationPending)
{
   // Das DownloadCompleted-Ereignis aufrufen
   asyncOperation.Post(this.downloadCompletedCallerHandler,
```

Listing 393: Dritter Teil der DownloadInternal-Methode

```
            new DownloadCompletedEventArgs(uri, null, true,
            destinationStream, asyncOperation.UserSuppliedState));

         // und raus
         return;
      }

      // Die Antwort anfordern und die Länge der Daten ermitteln
      try
      {
         response = (HttpWebResponse)request.GetResponse();
      }
      catch (WebException ex)
      {
         // Fehler beim Öffnen der Verbindung:
         // In der Regel ist der Webserver gerade nicht erreichbar:
         // Download erneut und weiter ausführen, wenn die maximale
         // Anzahl Versuche noch nicht erreicht ist
         if (downloadAttemptNumber >= maxDownloadAttempts)
         {
            // Maximalanzahl der Versuche wurde erreicht:
            // Das DownloadCompleted-Ereignis aufrufen
            WebException webException = new WebException(
               "Abbruch des Downloads nach " +
               downloadAttemptNumber + " erfolglosen Versuchen. " +
               "Letzter Fehler: " + ex.Message, ex);

            if (asyncOperation != null)
            {
               // Das DownloadCompleted-Ereignis aufrufen
               asyncOperation.Post(this.downloadCompletedCallerHandler,
                  new DownloadCompletedEventArgs(uri, webException,
                  false, destinationStream,
                  asyncOperation.UserSuppliedState));

               // und raus
               return;
            }
            else
            {
               // Busy-Flag zurücksetzen
               this.isBusy = false;

               // Exception werfen
               throw webException;
```

Listing 393: Dritter Teil der DownloadInternal-Methode (Forts.)

```
            }
        }

        // Pause zwischen den Downloadversuchen einlegen
        Thread.Sleep(this.delayBetweenDownloadAttempts);

        // while-Schleife an der nächsten Iteration ausführen
        continue;
    }

    // Einlesen der Anzahl der zu empfangenden Bytes
    totalBytesToReceive = response.ContentLength;
```

Listing 393: Dritter Teil der DownloadInternal-Methode (Forts.)

Nun beginnt das Lesen der Daten. Dazu ermittelt `DownloadInternal` über die Methode `GetResponseStream` des `HttpWebResponse`-Objekts den Antwort-Stream und liest diesen in einer inneren `do`-Schleife blockweise ein. Die Größe der auszulesenden Blöcke wird aus der Eigenschaft `BlockSize` ermittelt. Natürlich wird in der inneren Schleife wieder auf eine Abbruch-Anforderung reagiert, bevor die Daten über die `Read`-Methode des Antwort-Streams in einen Byte-Puffer und von da in den Ergebnis-Stream geschrieben werden.

Beim Lesen des Antwort-Streams kann es vorkommen, dass der Webserver (oder ein zwischengeschalteter Server oder Router) die Verbindung unterbricht. Dabei wird eine `IOException` generiert. Diese Exception fange ich ab. Dabei überprüft das Programm, ob die Maximal-Anzahl der Download-Versuche bereits erreicht ist. Ist dies der Fall, wird wieder eine `WebException` erzeugt und weitergegeben. Ist die Maximal-Anzahl noch nicht erreicht, wird die äußere Schleife (die `while`-Schleife) einfach dadurch wiederholt ausgeführt, dass die innere Schleife abgebrochen wird. Das Programm baut dann eine erneute Verbindung zum Webserver auf und beginnt den Download ab der Position, die den bereits gelesenen Daten entspricht, erneut. Vor dem Abbruch der Schleife wird allerdings noch eine kleine Pause eingelegt, deren Länge über die Eigenschaft `DelayBetweenDownloadAttempts` bestimmt ist. Unterhalb der inneren (`do`-)Schleife wird die äußere Schleife schließlich abgebrochen, wenn genügend Bytes gelesen wurden.

```
// Den Antwort-Stream ermitteln, diesen blockweise lesen
// und in den Ziel-Stream schreiben
responseStream = response.GetResponseStream();
int bytesRead = 0;
byte[] buffer = new byte[this.BlockSize];
do
{
    // Auf eine Abbruch-Anforderung reagieren
    if (asyncOperation != null && this.cancellationPending)
    {
```

Listing 394: DownloadInternal-Methode, Teil 4

```
            // Streams schließen
            responseStream.Close();
            response.Close();

            // Das DownloadCompleted-Ereignis aufrufen
            asyncOperation.Post(this.downloadCompletedCallerHandler,
               new DownloadCompletedEventArgs(uri, null, true,
               destinationStream, asyncOperation.UserSuppliedState));

            // und raus
            return;
         }

         try
         {
            bytesRead = responseStream.Read(
               buffer, 0, this.BlockSize);
         }
         catch (IOException ex)
         {
            // Fehler beim Lesen des Stream: In der Regel wurde
            // die HTTP-Verbindung vom Server geschlossen, weil
            // zu viele Daten gelesen wurden: Download erneut
            // und weiter ausführen, wenn die maximale Anzahl
            // Versuche noch nicht erreicht ist
            if (downloadAttemptNumber >= maxDownloadAttempts)
            {
               // Maximalanzahl der Versuche wurde erreicht:
               // Das DownloadCompleted-Ereignis aufrufen
               WebException webException = new WebException(
                  "Abbruch des Downloads nach " +
                  downloadAttemptNumber + " erfolglosen Versuchen. " +
                  "Letzter Fehler: " + ex.Message, ex);

               // Das DownloadCompleted-Ereignis aufrufen
               if (asyncOperation != null)
               {
                  asyncOperation.Post(
                     this.downloadCompletedCallerHandler,
                     new DownloadCompletedEventArgs(uri, webException,
                     false, destinationStream,
                     asyncOperation.UserSuppliedState));
               }
               else
               {
```

Listing 394: DownloadInternal-Methode, Teil 4 (Forts.)

```
                // Busy-Flag zurücksetzen
                this.isBusy = false;

                // Exception werfen
                throw webException;
            }
         }

         bytesRead = 0;

         // Pause zwischen den Downloadversuchen einlegen
         Thread.Sleep(this.delayBetweenDownloadAttempts);

         // do-Schleife abbrechen
         break;
      }

      // Fortschritt berechnen
      if (totalBytesToReceive > 0)
      {
         progressPercentage = (int)((totalBytesReceived /
            (float)totalBytesToReceive) * 100);
      }

      // Gelesene Bytes erhöhen und in den
      // Ergebnis-Stream schreiben
      if (bytesRead > 0)
      {
         totalBytesReceived += bytesRead;
         destinationStream.Write(buffer, 0, bytesRead);

         if (asyncOperation != null)
         {
            // Fortschritt melden
            asyncOperation.Post(
               this.downloadProgressChangedCallerHandler,
               new DownloadProgressChangedEventArgs(uri,
               totalBytesReceived, totalBytesToReceive,
               progressPercentage, DownloadState.ReadingData,
               downloadAttemptNumber, maxDownloadAttempts,
               asyncOperation.UserSuppliedState));
         }
      }
   } while (bytesRead > 0);
```

Listing 394: DownloadInternal-Methode, Teil 4 (Forts.)

```
            // Äußere Schleife abbrechen, wenn genügend Bytes gelesen wurden
            if (totalBytesReceived >= totalBytesToReceive)
            {
               break;
            }
         } // Ende der äußeren Schleife
```

Listing 394: DownloadInternal-Methode, Teil 4 (Forts.)

Der Abschluss der `DownloadInternal`-Methode ruft dann nur noch das `DownloadCompleted`-Ereignis auf, reagiert auf aufgetretene Exceptions und schließt die verwendeten Objekte:

```
if (asyncOperation != null)
   {
      // Das DownloadCompleted-Ereignis aufrufen
      asyncOperation.Post(this.downloadCompletedCallerHandler,
         new DownloadCompletedEventArgs(uri, null, false,
         destinationStream, asyncOperation.UserSuppliedState));
   }
}
catch (Exception ex)
{
   if (asyncOperation != null)
   {
      // Das DownloadCompleted-Ereignis aufrufen
      asyncOperation.Post(this.downloadCompletedCallerHandler,
         new DownloadCompletedEventArgs(uri, ex, false,
         destinationStream, asyncOperation.UserSuppliedState));
   }
   else
   {
      // Busy-Flag zurücksetzen
      this.isBusy = false;
      // Exception weiterwerfen
      throw ex;
   }
}
finally
{
   try
   {
      // Den Response-Stream schließen
      responseStream.Close();
   }
   catch { }
```

Listing 395: Abschluss der DownloadInternal-Methode

```
            try
            {
               // Das WebResponse-Objekt schließen
               response.Close();
            }
            catch { }
         }
      }
```

Listing 395: Abschluss der DownloadInternal-Methode (Forts.)

Die Methode `Download` führt nun einen synchronen Download durch, indem sie die `DownloadInternal`-Methode direkt aufruft. Da bei einem synchronen Download kein `AsyncOperation`-Objekt verwendet wird, übergibt `Download` an diesem Argument `null`. Um einen Mehrfach-Aufruf zu verhindern wird das Busy-Flag entsprechend abgefragt:


```
   public void Download(Uri uri, Stream destinationStream,
      byte maxDownloadAttempts)
   {
      // Überprüfen, ob gerade eine asynchrone Operation ausgeführt wird
      if (this.isBusy)
      {
         throw new InvalidOperationException(
            "DownloadAsync kann nicht aufgerufen werden " +
            "während ein Download ausgeführt wird");
      }

      try
      {
         // Das Busy-Flag setzen
         this.isBusy = true;

         // Download synchron starten
         this.DownloadInternal(uri, destinationStream,
            maxDownloadAttempts, null);
      }
      finally
      {
         // Das Busy-Flag zurücksetzen
         this.isBusy = false;
      }
   }
```

Listing 396: Synchroner Download einer Datei

Die Methode `DownloadAsync` lädt die Daten asynchron herunter. Die erste Variante dieser Methode (der kein Objekt für Benutzerdaten übergeben wird) ruft lediglich die zweite Variante

auf. Diese überprüft natürlich, ob das Busy-Flag gerade gesetzt ist, und wirft in diesem Fall eine `InvalidOperationException`. Im anderen Fall wird dieses Flag auf `true` gesetzt, ein `AsyncOperation`-Objekt erzeugt und die Arbeitsmethode asynchron aufgerufen.

```
/* Lädt eine Datei asynchron herunter */
public void DownloadAsync(Uri uri, Stream destinationStream,
  byte maxDownloadAttempts)
{
   this.DownloadAsync(uri, destinationStream, maxDownloadAttempts, null);
}

/* Lädt eine Datei asynchron herunter */
public void DownloadAsync(Uri uri, Stream destinationStream,
   byte maxDownloadAttempts, object userState)
{
   // Überprüfen, ob gerade eine asynchrone Operation ausgeführt wird
   if (this.isBusy)
   {
      throw new InvalidOperationException(
         "DownloadAsync kann nicht aufgerufen werden " +
         "während ein Download ausgeführt wird");
   }

   // Busy-Flag setzen
   this.isBusy = true;

   // AsyncOperation-Objekt erzeugen, über das die Ereignis-Handler
   // im richtigen Thread aufgerufen werden können
   AsyncOperation asyncOperation =
      AsyncOperationManager.CreateOperation(userState);

   // Die Arbeitsmethode asynchron aufrufen
   DownloadInternalCallerHandler workerEventHandler =
      new DownloadInternalCallerHandler(this.DownloadInternal);
   workerEventHandler.BeginInvoke(uri, destinationStream,
      maxDownloadAttempts, asyncOperation, null, null);
}
```

Listing 397: Methode zum asynchronen Download einer Datei

Die Methode `CancelAsync` ermöglicht schließlich den Abbruch eines (asynchronen) Downloads:

```
public void CancelAsync()
{
```

Listing 398: Methode zum Abbrechen eines Downloads und schließende Klammer der HttpDownload-Klasse

```
        this.cancellationPending = true;
    }
}
```

Listing 398: Methode zum Abbrechen eines Downloads und schließende Klammer der HttpDownload-Klasse (Forts.)

Die Anwendung der `HttpDownload`-Klasse ist nun recht einfach. Ich beschreibe diese kurz am Beispiel einer Windowsanwendung (Abbildung 122).

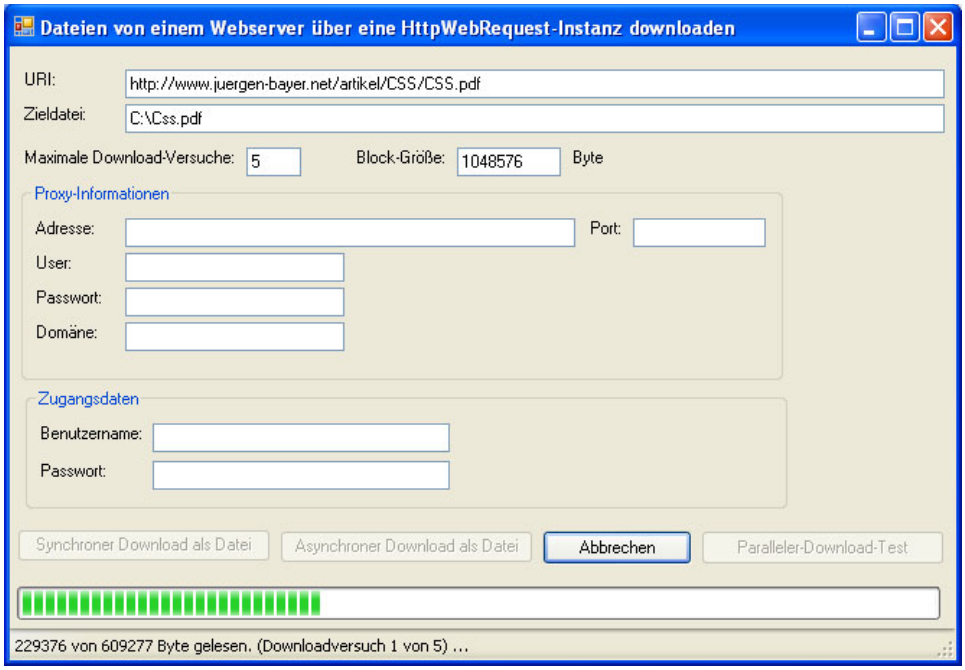


Abbildung 122: Beispielanwendung der HttpDownload-Klasse während des Downloads einer Datei

In der Formularklasse habe ich zunächst ein Feld implementiert, das die `HttpDownload`-Instanz verwaltet. Die Methode `UpdateUI` übernimmt die Aktualisierung der Benutzeroberfläche, die Methode `SetState` übernimmt die Umschaltung des Status.

```
/* HttpDownload-Referenz */
private HttpDownload httpDownload;

/* Aktualisiert die Benutzeroberfläche */
private void UpdateUI(string status, int progressPercentage)
```

Listing 399: HttpDownload-Referenz und Methoden zur Aktualisierung der Benutzeroberfläche

```
{
   this.toolStripStatusLabel.Text = status;
   this.progressBar.Value = progressPercentage;
}

/* Schaltet den Status des Formulars um */
private void SetState(bool downloading)
{
   this.cancelButton.Enabled = downloading;
   this.syncDownloadButton.Enabled = !downloading;
   this.asyncDownloadButton.Enabled = !downloading;
   this.parallelDownloadTestButton.Enabled = !downloading;
   this.progressBar.Value = 0;
}
```

Listing 399: HttpDownload-Referenz und Methoden zur Aktualisierung der Benutzeroberfläche (Forts.)

Dann folgen die Methoden für die Ereignisse der `HttpDownload`-Klasse. Die Methode `DownloadProgressChanged` fragt den jeweiligen Status ab und aktualisiert mit den übergebenen Daten über `UpdateUI` ein `StatusLabel` und eine `ProgressBar`. Die Methode `DownloadEnd` schließt den Stream, der in den Methoden für das `Click`-Ereignis der Download-Schalter geöffnet wurde. Die Methode `DownloadError` gibt die Fehlermeldung aus und schließt den Stream.

```
/* Methode für den Fortschritts-Delegaten */
private void DownloadProgressChanged(object sender,
   DownloadProgressChangedEventArgs e)
{
   lock (this)
   {
      switch (e.DownloadState)
      {
         case DownloadState.OpeningConnection:
            this.UpdateUI("Öffne die Verbindung. (Downloadversuch " +
               e.DownloadAttemptNumber + " von " +
               e.MaxDownloadAttempts + ") ... ", 0);
            break;

         case DownloadState.ReadingData:
            string status = e.BytesReceived + " von " +
               e.TotalBytesToReceive + " Bytes gelesen. (Downloadversuch " +
               e.DownloadAttemptNumber + " von " +
               e.MaxDownloadAttempts + ") ... ";
            this.UpdateUI(status, e.ProgressPercentage);
            break;
```

Listing 400: Methoden für die Ereignisse der HttpDownload-Klasse

```
      }
   }
}

/* Methode für den Download-Ende-Delegaten  */
private void DownloadCompleted(object sender, DownloadCompletedEventArgs e)
{
   // Die Datei schließen
   try
   {
      e.DestinationStream.Close();
   }
   catch (Exception ex)
   {
      MessageBox.Show("Fehler beim Schließen der Datei: " + ex.Message,
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }

   // Info ausgeben
   if (e.Cancelled)
   {
      this.UpdateUI("Abgebrochen", 0);
   }
   else if (e.Error != null)
   {
      MessageBox.Show(e.Error.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   else
   {
      this.UpdateUI("Fertig", 0);
   }

   // Abbrechen-Schalter deaktivieren und die anderen
   // Schalter aktivieren
   this.SetState(false);
}
```

Listing 400: Methoden für die Ereignisse der HttpDownload-Klasse (Forts.)

Die Methoden für die `Click`-Ereignisse der Download-Schalter rufen den synchronen bzw. asynchronen Download auf. Da der Programm-Quelltext bis auf den Methodennamen identisch ist, zeige ich lediglich die Methode zum asynchronen Download:

```
private void asyncDownloadButton_Click(object sender, System.EventArgs e)
{
   FileStream fileStream = null;
   try
   {
      // FileStream für die Datei erzeugen
      fileStream = new FileStream(this.destinationFilenameTextBox.Text,
         FileMode.Create, FileAccess.Write);
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);

      // FileStream schließen
      try
      {
         fileStream.Close();
      }
      catch { }

      return;
   }

   // Datei asynchron herunterladen
   try
   {
      // Status des Formulars setzen
      this.SetState(true);

      // HttpDownload-Instanz erzeugen
      this.httpDownload = new HttpDownload();

      // Proxy-Informationen übergeben
      if (this.proxyAddressTextBox.Text != null &&
         this.proxyAddressTextBox.Text.Length > 0)
      {
         int proxyPort = 0;
         if (this.proxyPortTextBox.Text != null &&
            this.proxyPortTextBox.Text.Trim().Length > 0)
         {
            proxyPort = Convert.ToInt32(this.proxyPortTextBox.Text);
         }
```

Listing 401: Asynchroner Download

```
            this.httpDownload.Proxy =
               new WebProxy(this.proxyAddressTextBox.Text,
               proxyPort);
            this.httpDownload.Proxy.Credentials = new NetworkCredential(
               this.proxyUsernameTextBox.Text, this.proxyPasswordTextBox.Text,
               this.proxyUserDomainTextBox.Text);
         }

         // Zugangsdaten übergeben
         if (this.usernameTextBox.Text != null &&
            this.usernameTextBox.Text.Length > 0)
         {
            this.httpDownload.Credentials = new NetworkCredential(
               this.usernameTextBox.Text, this.passwordTextBox.Text);
         }

         // Ereignisse zuweisen
         this.httpDownload.DownloadProgressChanged += new
            Addison_Wesley.Codebook.Internet.
            DownloadProgressChangedEventHandler(
            this.DownloadProgressChanged);
         this.httpDownload.DownloadCompleted +=
            new DownloadCompletedEventHandler(this.DownloadCompleted);

         // Die Block-Größe übergeben
         this.httpDownload.BlockSize =
            Convert.ToInt32(this.blockSizeTextBox.Text);

         // Die Datei asynchron herunterladen
         this.httpDownload.DownloadAsync(new Uri(this.uriTextBox.Text),
            fileStream, Convert.ToByte(this.maxDownloadAttemptsTextBox.Text));
      }
      catch (Exception ex)
      {
         this.SetState(false);
         MessageBox.Show(ex.Message, Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Error);
      }
   }
```

Listing 401: Asynchroner Download (Forts.)

Die Ereignismethode des Schalters zum Abbrechen des Downloads schließlich ruft lediglich die `CancelAsync`-Methode der `HttpDownload`-Instanz auf. Diese Methode funktioniert natürlich nur dann, wenn der Download in einem Thread ausgeführt wird.

```
private void cancelButton_Click(object sender, EventArgs e)
{
   this.httpDownload.CancelAsync();
}
```

Listing 402: Abbruch eines Downloads

Achtung

Meine Download-Klasse ist nicht ganz perfekt: Wenn der Webserver den Download kurz unterbricht und zwischen der Unterbrechung und der Wiederaufnahme die herunterzuladende Datei auf dem Webserver geändert wird, sind die heruntergeladenen Daten korrupt. Dieses Problem könnte gelöst werden, indem vor jeder Wiederaufnahme des Downloads das Dateidatum und die Dateigröße abgefragt und mit den Daten des ersten Downloadversuchs verglichen werden. Zur Implementierung dieses Features hatte ich aber leider keine Zeit mehr. Falls Sie das Problem lösen, würde ich mich über eine E-Mail mit der Lösung freuen ☺.

Tipp

Sie können den Download unter erschwerten Bedingungen testen, indem Sie eine große Datei (mehrere MB) von dem lokalen Webserver herunterladen und die Website zwischenzeitlich ein paar Mal kurz anhalten. Der Download sollte in diesem Fall bis zur angegebenen Versuchs-Maximalanzahl immer wieder erneut an der letzten Position starten.

Tipp

Die `HttpWebRequest`-Klasse ermöglicht auch den Download von Dateien über eine SSL-Verbindung (über eine https-Adresse, siehe bei *de.wikipedia.org/wiki/Transport_Layer_Security*). Das bei einer solchen (geschützten) Verbindung vom Server zu seiner Authentifizierung gesendete Zertifikat muss entweder eines sein, das von einer der bekannten Zertifizierungsstellen stammt oder für das auf dem lokalen Rechner eine Vertrauensstellung eingerichtet ist. Der Versuch, über eine SSL-Verbindung mit einem ungültigen (nicht zur Web-Domäne gehörenden, abgelaufenen etc.) Zertifikat eine Datei herunterzuladen, schlägt mit dem folgenden Fehler fehl: »Die zugrunde liegende Verbindung wurde geschlossen: Für den geschützten SSL/TLS-Kanal konnte keine Vertrauensstellung hergestellt werden«. Die innere Exception meldet den Fehler »Das Remotezertifikat ist laut Validierungsverfahren ungültig.«. Diese Exception ist absolut gerechtfertigt, da das Zertifikat ein wichtiger Beweis dafür ist, dass es sich beim Webserver tatsächlich um den angesprochenen Server handelt. Der Internet Explorer meldet beim Versuch, von einem solchen Webserver über SSL eine Datei herunterzuladen, dass das Zertifikat ungültig ist, und ermöglicht es, die Datei trotzdem herunterzuladen. Dieses Rezept könnte entsprechend erweitert werden, aber bisher habe ich dafür noch keine Lösung gefunden.

Tipp

Den Zugriff über einen Proxy-Server können Sie sehr gut über den Freeware-Server *Sambar* (*www.sambar.de*) testen. Sambar ist neben einem WebServer auch ein Proxy-Server, der sehr einfach zu konfigurieren ist und der auch problemlos zum Test (zum Beispiel des authentifizierten Zugriffs) verwendet werden kann, wenn der Internetzugriff über eine direkte LAN- oder DSL-Verbindung erfolgt. Achten Sie in der Konfiguration gegebenenfalls darauf, dass Sie den HTTP-Port auf einen anderen Wert als den voreingestellten Port 80 (der standardmäßig vom IIS verwendet wird) einstellen (SERVER / HTTP / SERVER PORT). Unter SERVER / ÜBERBLICK können Sie einstellen, dass Sambar als HTTP-Proxy arbeitet. Außerdem können Sie hier festlegen, dass der Proxy eine Authentifizierung erzwingt (PROXY AUTHENTICATION). Die Benutzer, die sich gegenüber Sambar anmelden können, verwalten Sie über das »Menü« SICHERHEIT / BENUTZER. Stellen Sie gegebenenfalls noch ein, dass der HTTP-Proxy-Zugriff protokolliert wird, damit Sie diesen überprüfen können (SERVER / PROXIES / LOG HTTP PROXY USAGE).

Hinweis

In Ihren Programmen sollten Sie für den Zugriff über einen Proxy-Server beachten, dass die Klasse `HttpWebRequest` grundsätzlich die im Internet Explorer eingestellten Proxy-Einstellungen verwendet, wenn keine spezifischen Proxy-Einstellungen angegeben werden.

Der asynchrone Download funktionierte in meinem Test einwandfrei. Es war sogar problemlos möglich, (mit mehreren `HttpDownload`-Instanzen) mehrere Downloads parallel auszuführen, natürlich nur, so lange der Webzugriff nicht durch eine Firewall oder Ähnliches verhindert wurde. Bei Ihren Versuchen mit asynchronen Downloads sollten Sie darauf achten, die Blockgröße nicht zu klein einzustellen. Ist diese z. B. auf 1024 eingestellt, erfolgt das Aktualisieren der Benutzeroberfläche in der Regel in so kurzen Abständen, dass Sie nicht allzu viel davon merken, dass der Download asynchron ausgeführt wird.

217 Intelligenter Up- und Download mit BITS

BITS (Background Intelligent Transfer Service) ist ein Datei-Übertragungsdienst, der ab Windows XP per Default zur Verfügung steht und unter Windows 2000 nachinstalliert werden kann. In der deutschen Version von Windows XP oder Server 2003 wird dieser Dienst als »Intelligenter Hintergrundübertragungsdienst« bezeichnet. Über BITS können Dateien im Hintergrund von einem Webserver über HTTP oder HTTPS heruntergeladen oder zu einem Server hochgeladen werden. Das automatische Windows-Update arbeitet z. B. mit dieser Technologie.

Der Down- bzw. Upload ist dabei einigermaßen intelligent. Die Datenübertragung erfolgt unter Ausnutzung der optimalen Bandbreite des Netzwerks. Ist das Netzwerk z. B. zurzeit mit 60% ausgenutzt, verwendet BITS lediglich die freien 40% der Bandbreite. Wird während der Datenübertragung die Serververbindung unterbrochen, versucht BITS es zu einem späteren Zeitpunkt einfach erneut, wobei der Download bzw. Upload an der letzten Position weiter ausgeführt wird. Ähnliches geschieht, wenn der Benutzer sich zwischenzeitlich abmeldet bzw. den Rechner herunterfährt. Der Download wird dann so lange unterbrochen, bis der Benutzer sich wieder anmeldet.

Bei der Wiederaufnahme der Datenübertragung wird überprüft, ob die Größe der Datei oder das Dateidatum sich zwischenzeitlich verändert haben. Ist dies der Fall, wird die Datenübertragung einfach wieder von vorn ausgeführt. Damit verhindert BITS, dass Dateien bei einer Unterbrechung eines Ladevorgangs ungültig werden.

BITS wird vom Betriebssystem verwaltet. Ein Programm kann dem Dienst Jobs hinzufügen, muss aber während der Datenübertragung nicht weiter ausgeführt werden. Das Betriebssystem sorgt dafür, dass die Dateien übertragen werden. Ein Programm kann einen Job erzeugen, beendet werden und später, nach einem Neustart, den Status der laufenden Jobs überprüfen.

BITS 1.0 ist in Windows XP integriert. Diese Version erlaubt nur den Download von Dateien. Die Version 1.5, die mit Windows Server 2003 mitgeliefert wird, ermöglicht auch den Upload. Allerdings muss dazu der IIS ab der Version 5.0 mit installierter BITS-Server-Erweiterung eingesetzt werden. Das Service Pack 2 von Windows XP installiert BITS in der Version 2.0. Die aktuelle Version können Sie über die BITS-Startseite (*msdn.microsoft.com/en-us/library/bb968799(VS.85).aspx*) bei Microsoft herunterladen. Diese Version können Sie auch auf Windows 2000 SP3 installieren. Auf der BITS-Seite finden Sie auch einige Artikel über diese Technologie.

Welche Version auf einem System installiert ist, können Sie über das *BITS Admin Tool* herausfinden. Diese Kommandozeilen-Anwendung ist Bestandteil des Windows-SDK, kann aber auch bei Microsoft direkt heruntergeladen werden. Das Problem ist lediglich (wie so oft bei Microsoft) den Download zu finden. Suchen Sie bei *www.microsoft.com/downloads/Search.aspx* nach »Windows Server 2003 Service Pack 2 32-bit Support Tools«. Die Datei *support.cab*, die Sie mit WinRar (*www.winrar.de*) entpacken können, enthält die Datei *bitsadmin.exe*. Dies ist das BITS Admin Tool.

Wenn Sie an der Kommandozeile *bitsadmin* aufrufen, erhalten Sie eine Information über die Version. Ein Aufruf mit dem Schalter /? zeigt Informationen über die Möglichkeit dieses Tools an.

BITS ist zurzeit noch nicht in das .NET Framework integriert. BITS besteht aus COM-Komponenten, deren Handhabung leider nicht allzu einfach ist. Aber dieses Problem können Sie vernachlässigen, denn mittlerweile existieren Wrapper-Assemblys für die COM-Komponenten. Microsoft selbst stellte ein Visual-Basic-Projekt im MSDN zur Verfügung (das aber scheinbar nicht mehr existiert). Die Klassen in dieser Komponente waren aber in Bezug auf die Ereignisbehandlung nicht besonders ausgereift. Die Komponente von David Hall, die Sie an der Adresse *code.msdn.microsoft.com/bits* finden, ist in Bezug auf die Ereignisse wesentlich einfacher anzuwenden und besteht in der kompilierten Form aus nur einer Assembly. Sie finden diese Komponente in einer von mir modifizierten Version im Ordner *Komponenten* auf der Buch-CD. Davids Komponente hat übrigens sehr viel Ähnlichkeit mit dem Microsoft-Beispiel.

Die aktuelle Version finden Sie (hoffentlich noch) an der Adresse *code.msdn.microsoft.com/bits*. Die Seite *www.gotdotnet.com* ist aber bereits abgelaufen, weswegen der Link bald nicht mehr funktionieren wird.

Hinweis

Die zum Zeitpunkt der ersten Erstellung diese Beispiels aktuelle BITS-Komponente von David Hall war noch nicht ganz fehlerfrei und vollständig: BITS ermöglicht die Auflistung der aktuellen Jobs. Davids Klasse listet immer alle Jobs auf. Das führt zu einer Exception, wenn der aktuelle Benutzer kein Administrator ist, denn nur Administratoren dürfen alle Jobs auflisten. Andere Benutzer können nur ihre eigenen Jobs auflisten und bearbeiten. Ich habe die Klasse `JobList` so abgeändert, dass nur dann alle Jobs aufgelistet werden, wenn ein Administrator eingeloggt ist. Ist ein anderer Benutzer eingeloggt, werden nun nur dessen Jobs aufgelistet. Außerdem fehlt den Klassen von David Hall die Möglichkeit, den letzten Fehler für einen Job abzufragen, falls dieser fehlerhaft beendet wurde. Deshalb habe ich die Methode `GetLastError` hinzugefügt, die den letzten Fehler in Form einer `BitsError`-Instanz zurückgibt. Ich habe die Änderungen David gemailt. Mittlerweile wurde die BITS-Komponente aktualisiert, aber die `GetLastError`-Methode und die Einschränkung der Jobs fehlte (laut des Quellcodes) immer noch. Da meine Version sehr gut funktioniert und ich ehrlich gesagt keine Lust hatte, wieder alles erneut auszuprobieren, habe ich meine Version beibehalten.

Das Prinzip von BITS sind Jobs, die über das Admin-Tool oder über das BITS-API hinzugefügt und gesteuert werden können. Ein Job kann mehrere Dateien übertragen und ist mit einer bestimmten Priorität ausgestattet. Eine Anwendung fügt BITS einen Job hinzu und wartet dann darauf, dass dieser erfolgreich beendet wurde. Dazu kann die Anwendung den Status des Jobs abfragen oder auf Ereignisse reagieren. BITS überträgt die Dateien der Reihe nach, also nicht parallel.

Jobs werden im Kontext des aktuellen Benutzers ausgeführt. Meldet sich Benutzer A ab und Benutzer B an, werden die Jobs von Benutzer A so lange angehalten, bis er sich wieder anmeldet. Benutzer dürfen nur ihre eigenen Jobs auflisten und bearbeiten, mit Ausnahme von Administratoren, die alle Jobs bearbeiten können. Jobs von Benutzern, die sich nie wieder anmelden, werden nach 90 Tagen automatisch entfernt.

BITS erfordert keine besonderen Rechte. Selbst Benutzer der Benutzer-Gruppe von Windows, die mit nur minimalen Rechten ausgestattet sind, dürfen über BITS Dateien hoch- und herunterladen. Dazu muss der Benutzer natürlich Lese- und gegebenenfalls Schreibrechte auf der lokalen Datei bzw. dem lokalen Ordner besitzen.

Zur Arbeit mit BITS referenzieren Sie die Assembly *Microsoft.Net.BITS.dll* oder integrieren Sie das Visual-Basic-Projekt in die Projektmappe und referenzieren Sie dieses. Alle verwendeten Klassen werden im Namensraum `Microsoft.Net.BITS` verwaltet.

Hinweis

Ich bin mir nicht sicher, ob es wichtig ist, aber einige Probleme mit den BITS-Ereignissen haben mich vorsichtig werden lassen: Achten Sie darauf, dass die `Main`-Methode der Anwendung mit dem `STAThread`-Attribut belegt ist. Damit sorgen Sie dafür, dass die Anwendung für COM ein *Single Threaded Apartment* (STA) einsetzt. Viele COM-Komponenten sind so geschrieben, dass diese nur in einem STA korrekt arbeiten. BITS gehört wahrscheinlich dazu.

Zum Hinzufügen eines Jobs rufen Sie die statische Methode `CreateJob` der `BackgroundCopyManager`-Klasse auf. Dieser Methode können Sie einen Anzeige-Namen und eine Beschreibung

für den Job übergeben. Am dritten Argument der in diesem Rezept verwendeten Variante übergeben Sie einen Wert der `BackgroundCopyJobType`-Aufzählung, der bestimmt, ob Sie einen Upload oder einen Download ausführen wollen. Die dritte Möglichkeit hier heißt `UploadReply` (Upload-Rückantwort) und ist wahrscheinlich als Antwort auf einen Upload gedacht.

`CreateJob` gibt eine Instanz der Klasse `BackgroundCopyJob` zurück, über die Sie weiter mit dem Job arbeiten. So können Sie z. B. die Ereignisse `Changed`, `Completed` und `Error` zuweisen, die aufgerufen werden, wenn sich an dem Status des Jobs etwas ändert, der Job erfolgreich ausgeführt wurde oder ein Fehler aufgetreten ist.

Über die `Add`-Methode der `Files`-Eigenschaft können Sie dann einzelne Dateien hinzufügen. Am ersten Argument übergeben Sie den URI der entfernten Datei, am zweiten den Dateinamen der lokalen Datei.

Die Eigenschaft `Priority` bestimmt die Priorität des Jobs. Hier können Sie die folgenden Werte der `BackgroundCopyJobPriority`-Aufzählung verwenden: `Foreground`, `High`, `Low` und `Normal`. Die Priorität `Foreground` ist die höchste und bestimmt, dass der Job unabhängig von der verfügbaren Netzwerk-Bandbreite so schnell wie möglich ausgeführt wird. Die anderen Prioritäten berücksichtigen die verfügbare Bandbreite. Die Priorität eines Jobs bestimmt auch den Zeitpunkt, zu dem dieser gestartet wird.

Falls der Zugang zum Internet über einen Proxy-Server erfolgt, können Sie in die Eigenschaft `Proxy` des `BackgroundCopyJob`-Objekts eine Instanz der `WebProxy`-Klasse schreiben, die entsprechend initialisiert ist.

Achtung

Dabei sollten Sie jedoch beachten, dass die `Credentials`-Eigenschaft der `WebProxy`-Instanz von der `BackgroundCopyJob`-Instanz (bzw. von BITS) nicht verwendet wird. Falls der Proxy-Server eine Authentifizierung verlangt, werden die dafür notwendigen Benutzerdaten nicht übergeben, auch wenn Sie `Credentials` entsprechend belegen. U. U. können Sie die Authentifizierungsdaten über die `SetCredentials`-Methode des Job übergeben (so suggeriert es jedenfalls die Beschreibung dieser Methode), aber das konnte ich leider nicht testen, da BITS meinen lokalen Proxy nicht erkannte.

Sie sollten außerdem beachten, dass BITS automatisch die im Internet Explorer eingestellten Proxy-Informationen verwendet. Die `Proxy`-Eigenschaft sollten Sie also nur dann initialisieren, wenn sich die Proxy-Informationen unterscheiden.

Hinweis

In meinen Versuchen führte das Setzen der `Proxy`-Eigenschaft lediglich dazu, dass BITS für den jeweiligen Job den Fehler »Der angegebene Proxyserver bzw. die Umgehungsliste ist ungültig« meldete. Ein paralleler Versuch mit dem Microsoft-BITS-Wrapper führte zu einem ähnlichen Ergebnis: Hier wurde zwar kein Fehler erzeugt, der eingestellte Proxy-Server wurde in diesem Fall aber einfach ignoriert.

Für den Fall, dass der Server eine Authentifizierung verlangt, können Sie die dazu notwendigen Informationen über die Methode `SetCredentials` setzen, der Sie ein `NetworkCredential`-Objekt übergeben. Wahrscheinlich können Sie hier auch Authentifizierungsdaten für einen Proxy-Server übergeben. Dieses Feature habe ich allerdings nicht getestet.

Nachdem Sie einen Job erzeugt haben, befindet sich dieser zunächst im Warte-Status. Um den Job zu starten rufen Sie die `Resume`-Methode auf.

Das folgende Windows-Anwendungs-Beispiel, das den Import der Namensräume `System.Net` und `Microsoft.Net.BITS` erfordert, startet einen neuen Download-Job mit den URI- bzw. Dateiangaben, die in die Textboxen `uriTextBox` und `destinationFilenameTextBox` eingegeben wurden. Das Beispiel weist die Ereignisse `Changed`, `Completed` und `Error` auf passende Methode zu:

```
string uri = this.uriTextBox.Text;
string destinationFile = this.destinationFilenameTextBox.Text;
string displayName = "Download-Demo";
string description = "Download von '" + uri + "'";

// Job erzeugen
BackgroundCopyJob job = BackgroundCopyManager.CreateJob(displayName,
   description, BackgroundCopyJobType.Download);

// Ereignisse zuweisen
job.Changed += new BackgroundCopyJobEventHandler(this.job_Changed);
job.Completed += new BackgroundCopyJobEventHandler(this.job_Completed);
job.Error += new System.IO.ErrorEventHandler(this.job_Error);

job.Files.Add(uri, destinationFile);
job.Priority = BackgroundCopyJobPriority.Normal;
job.Resume();
```

Listing 403: Starten eines BITS-Download-Jobs

In der Ereignismethode für das `Changed`-Ereignis erhalten Sie in der Eigenschaft `Job` des Ereignisargument-Objekts eine Referenz auf den Job. Über diese Referenz können Sie alle Eigenschaften des Job abfragen. Wenn Sie im `Changed`-Ereignis Steuerelemente aktualisieren, müssen Sie allerdings beachten, dass das Ereignis in einem anderen (COM-)Thread ausgeführt wird. Deshalb dürfen Sie nicht auf Steuerelemente direkt zugreifen, sondern müssen dazu die `Invoke`-Methode des Steuerelements oder des Formulars (bzw. in einer WPF-Anwendung des `Dispatcher`-Objekts) verwenden (vergleichen Sie dazu das Rezept 365). Dazu benötigen Sie einen Delegaten und eine Methode. Das folgende Beispiel dient der Aktualisierung der `Text`-Eigenschaft eines Steuerelements:

```
private delegate void changeTextHandler(Control control, string newText);
private void ChangeText(Control control, string newText)
{
   control.Text = newText;
}
```

Listing 404: Delegat und Methode für die Aktualisierung eines Steuerelements

In der Methode für das `Changed`-Ereignis schreibt mein Beispiel den Anzeigenamen des Jobs und eine Information über die bisher übertragenen Bytes in das `Label infoLabel`:

```
private void job_Changed(object sender, BackgroundCopyJobEventArgs args)
{
   string info = "Job " + args.Job.DisplayName + ": " +
      args.Job.Progress.BytesTransferred + " von " +
      args.Job.Progress.BytesTotal + " geladen";
   this.infoLabel.Invoke(new changeTextHandler(this.ChangeText),
      new object[] { this. infoLabel, info });
}
```

Listing 405: Methode für das Changed-Ereignis der Jobs

Dem `Completed`-Ereignis wird dasselbe Ereignisargument-Objekt übergeben wie dem `Changed`-Ereignis. Mein Beispiel fragt hier lediglich den Anzeigennamen ab, um im Info-`Label` auszugeben, dass der Job fertig gestellt wurde:

```
private void job_Completed(object sender, BackgroundCopyJobEventArgs args)
{
   string info = args.Job.DisplayName + " ist fertig gestellt";
   this.infoLabel.Invoke(new changeTextHandler(this.ChangeText),
      new object[] { this.infoLabel, info });
}
```

Listing 406: Methode für das Completed-Ereignis

Das `Error`-Ereignis wird aufgerufen, wenn während der Dateiübertragung ein Fehler auftritt, dessen Ursache nicht repariert werden kann. In diesem Ereignis bekommen Sie leider keine Referenz auf den Job, sondern lediglich eine Referenz auf die aufgetretene `Exception` übergeben. Die Aussagekraft einer Fehlermeldung ist damit leider nicht allzu groß:

```
private void job_Error(object sender, System.IO.ErrorEventArgs e)
{
   string info = "Fehler: " + e.GetException().Message;
   this.infoLabel.Invoke(new changeTextHandler(this.ChangeText),
      new object[] { this.infoLabel, info });
}
```

Listing 407: Methode für das Error-Ereignis

Achtung

Bei meinen Versuchen wurden zumindest das `Changed`-Ereignis und das `Completed`-Ereignis in einer Windowsanwendung nur dann aufgerufen, wenn ich das Programm direkt über die ausführbare Datei gestartet hatte (`Error` wurde nie aufgerufen, weil kein Fehler auftrat). Bei einem Start über Visual Studio 2008 wurden diese Ereignisse in einer Windowsanwendung nie aufgerufen (allerdings in einer Konsolenanwendung). Ein Test mit dem Microsoft-BITS-Wrapper (den Sie neben dem Beispiel zu diesem Rezept finden) führte zu ähnlichen Ergebnissen: Hier wurde das `JobTransferred`-Ereignis ebenfalls nur aufgerufen, wenn das Programm direkt gestartet wurde (das `JobModification`-Ereignis wurde nie aufgerufen). Wahrscheinlich handelt es sich dabei um Inkompatibilitäten zwischen dem Visual-Studio-Debugger und COM.

Die Jobs, die Sie einmal gestartet haben, bleiben so lange halten, bis diese abgebrochen oder abgeschlossen wurden. Sie können dazu die `Cancel`- bzw. die `Completed`-Methode des jeweiligen `Job`-Objekts aufrufen. `Cancel` bricht, wie der Name schon sagt, die Ausführung des Jobs ab. `Complete` sagt dem System, dass der Job in Ihren Augen erfolgreich ausgeführt wurde. Erst nach dem Aufruf von `Complete` wird eine heruntergeladene Datei (die bis dahin in einem temporären Ordner verwaltet wurde) in die bei der Erzeugung des Jobs angegebene lokale Datei kopiert. Der Job wird dann auch aus der Job-Liste entfernt. Das Kopieren der Dateien und das Entfernen des Jobs geschehen nicht automatisch, damit ein Benutzer jederzeit den Status der von ihm gestarteten Jobs überprüfen kann.

Über die statische `Jobs`-Eigenschaft der `BackgroundCopyManager`-Klasse erreichen Sie alle aktuellen Jobs. Wie ich bereits angemerkt habe, habe ich die `JobList`-Klasse so abgeändert, dass nur dann alle Jobs aufgelistet werden, wenn ein Administrator eingeloggt ist. Ist ein Nicht-Administrator eingeloggt, werden nur die Jobs aufgelistet, die der Benutzer selbst gestartet hat.

Über die referenzierten `BackgroundCopyJob`-Objekte können Sie die Jobs abfragen und steuern. Jeder Job besitzt eine Id in Form einer GUID, die Sie aus der Eigenschaft `ID` auslesen können. Über diese GUID können Sie sich z. B. alle Jobs merken, die eine Anwendung gestartet hat.

Mein Beispiel schreibt die Ids der aktuellen Jobs lediglich in die `ListBox jobListBox`. Die Methode `ListJobs` merkt sich dabei den aktuell selektierten Job um diesen nach dem Neufüllen wieder zu selektieren:

```
private void ListJobs()
{
   // Info-Label leeren
   this.jobInfoLabel.Text = null;

   // Den aktuell selektierten Job merken
   Nullable<Guid> currentJob = null;
   if (this.jobListBox.SelectedIndex > -1)
   {
      currentJob = (Guid)this.jobListBox.SelectedItem;
```

Listing 408: Einlesen der IDs der aktuellen Jobs in eine ListBox

```
    }
    try
    {
       // Die aktuellen Jobs neu einlesen
       this.jobListBox.Items.Clear();
       foreach (BackgroundCopyJob job in BackgroundCopyManager.Jobs)
       {
          int index = this.jobListBox.Items.Add(job.ID);
          if (currentJob != null && currentJob.Value.Equals(job.ID))
          {
             // Den zuvor selektierten Job wieder selektieren
             this.jobListBox.SelectedIndex = index;
          }
       }
    }
    catch (Exception ex)
    {
       MessageBox.Show(ex.Message, Application.ProductName,
          MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Listing 408: Einlesen der IDs der aktuellen Jobs in eine ListBox (Forts.)

Über den Indexer der `Jobs`-Eigenschaft der `BackgroundCopyManager`-Klasse können Sie alle Jobs über deren Id oder Anzeigenamen referenzieren. Übergeben Sie dem Indexer eine GUID, sprechen Sie die Id an. Über das zurückgehaltene `BackgroundCopyJob`-Objekt können Sie Informationen über den Job einlesen und den Job steuern. Wie Sie ja bereits wissen, verwaltet die Eigenschaft `ID` die Id des Jobs, `DisplayName` verwaltet den Anzeigenamen und `Description` die Beschreibung.

Über die `Files`-Eigenschaft erreichen Sie `BackgroundCopyFileInfo`-Objekte mit Informationen zu den Dateien, die im Job übertragen werden.

Die Eigenschaft `State` gibt Auskunft über den Status. Die verfügbaren Stati entnehmen Sie bitte Listing 409. Dieses Beispiel schreibt Informationen über den in der `ListBox jobListBox` selektierten Job in das `Label jobInfoLabel`:

```
private void GetJobInfo()
{
   // Job referenzieren
   Guid jobId = new Guid(this.jobListBox.SelectedItem.ToString());
   BackgroundCopyJob job = BackgroundCopyManager.Jobs[jobId];
   string jobInfo = "Id: " + job.ID.ToString() + Environment.NewLine +
      "Anzeigename: " + job.DisplayName + Environment.NewLine +
      "Beschreibung: " + job.Description + Environment.NewLine +
```

Listing 409: Auslesen von Informationen über einen Job

```
      "Dateien: " + Environment.NewLine;

   // Dateien abfragen
   string files = null;
   foreach (BackgroundCopyFileInfo fileInfo in job.Files)
   {
      if (files != null)
      {
         files += Environment.NewLine;
      }
      files += "   " + fileInfo.RemoteFilePath +
         " -> " + fileInfo.LocalFilePath;
   }
   jobInfo += files + Environment.NewLine;

   // Status abfragen
   string jobState = null;
   switch (job.State)
   {
      case BackgroundCopyJobState.Transferring:
         jobState = "Dateien werden geladen: " +
            job.Progress.BytesTransferred +
            " von " + job.Progress.BytesTotal + " Bytes";
         break;

      case BackgroundCopyJobState.Cancelled:
         jobState = "Der Job wurde abgebrochen";
         break;

      case BackgroundCopyJobState.Connecting:
         jobState = "BITS verbindet mit dem Server";
         break;

      case BackgroundCopyJobState.Queued:
         jobState = "Der Job wurde in die " +
            "Warteschlange eingereiht";
         break;

      case BackgroundCopyJobState.Suspended:
         jobState = "Der Job ist ausgesetzt";
         break;

      case BackgroundCopyJobState.TransientError:
         jobState = "Ein transienter Fehler trat auf";
         break;
```

Listing 409: Auslesen von Informationen über einen Job (Forts.)

```
      case BackgroundCopyJobState.Error:
         BitsError error = job.GetLastError();
         if (error != null)
         {
            jobState = "Fehler " + error.Code +
               " bei der Ausführung des Jobs: " + error.Description +
               " (Kontext: " + error.ContextDescription + ")";
         }
         else
         {
            jobState = "Unbekannter Fehler";
         }
         break;

      case BackgroundCopyJobState.Transferred:
         jobState = "Der Job wurde " +
            "erfolgreich ausgeführt";
         break;
   }
   jobInfo += "Status: " + jobState + Environment.NewLine;

   // Fehler abfragen
   jobInfo += "Fehler bisher: " + job.ErrorCount;

   // Info im Label ausgeben
   this.jobInfoLabel.Text = jobInfo;
}
```

Listing 409: Auslesen von Informationen über einen Job (Forts.)

Jobs im Status `BackgroundCopyJobState.Transferred` wurden erfolgreich ausgeführt. Diese Jobs sollten Sie über den Aufruf der `Complete`-Methode abschließen, damit die heruntergeladene Datei in die Zieldatei kopiert wird. Der Status `BackgroundCopyJobState.Error` gibt an, dass ein Fehler aufgetreten ist, der nicht automatisch behandelt werden kann. Die Eigenschaft `ErrorCount` gibt eine Auskunft darüber, wie viele Fehler insgesamt aufgetreten sind. Über die (von mir hinzugefügte) Job-Methode `GetLastError` können Sie den letzten aufgetretenen Fehler für einen Job auslesen. Die Eigenschaft `Description` des von dieser Methode zurückgegebenen `BitsError`-Objekts liefert die Beschreibung des Fehlers. Über die Eigenschaft `Code` können Sie die (COM-)Fehlernummer auslesen. `Context` liefert eine Information über den Kontext des Fehlers und aus der Eigenschaft `ContextDescription` können Sie eine Beschreibung des Kontextes auslesen.

Über die `Suspend`-Methode können Sie laufende Jobs anhalten. Die `Resume`-Methode sorgt dafür, dass angehaltene Jobs weiter ausgeführt werden. Über `Cancel` brechen Sie ein Job ab.

Das folgende Beispiel reagiert auf die Betätigung verschiedener Schalter zum Anhalten, Weiter-Ausführen, Abbrechen und Abschließen des Jobs, der in der `ListBox jobListBox` gerade selektiert ist:

```
/* Hält den aktuellen Job an */
private void suspendButton_Click(object sender, EventArgs e)
{
   if (jobListBox.SelectedIndex > -1)
   {
      // Job referenzieren
      Guid jobId = new Guid(this.jobListBox.SelectedItem.ToString());
      BackgroundCopyJob job = BackgroundCopyManager.Jobs[jobId];

      // Job anhalten
      job.Suspend();

      // Jobs neu auflisten
      this.ListJobs();
   }
}

/* Führt den aktuellen Job weiter aus, wenn dieser angehalten ist */
private void resumeButton_Click(object sender, EventArgs e)
{
   if (jobListBox.SelectedIndex > -1)
   {
      // Job referenzieren
      Guid jobId = new Guid(this.jobListBox.SelectedItem.ToString());
      BackgroundCopyJob job = BackgroundCopyManager.Jobs[jobId];

      // Job weiter ausführen
      job.Resume();

      // Jobs neu auflisten
      this.ListJobs();
   }
}

/* Bricht den aktuellen Job ab */
private void cancelButton_Click(object sender, EventArgs e)
{
   if (jobListBox.SelectedIndex > -1)
   {
      // Job referenzieren
      Guid jobId = new Guid(this.jobListBox.SelectedItem.ToString());
      BackgroundCopyJob job = BackgroundCopyManager.Jobs[jobId];

      // Job abbrechen
      job.Cancel();
```

Listing 410: Methoden für Schalter zum Anhalten, weiter Ausführen, Abbrechen und Abschließen von Jobs

```
      // Jobs neu auflisten
      this.ListJobs();
   }
}

/* Schließt den aktuellen Job ab */
private void completeButton_Click(object sender, EventArgs e)
{
   if (jobListBox.SelectedIndex > -1)
   {
      // Job referenzieren
      Guid jobId = new Guid(this.jobListBox.SelectedItem.ToString());
      BackgroundCopyJob job = BackgroundCopyManager.Jobs[jobId];

      // Job abschließen
      job.Complete();

      // Jobs neu auflisten
      this.ListJobs();
   }
}
```

Listing 410: Methoden für Schalter zum Anhalten, weiter Ausführen, Abbrechen und Abschließen von Jobs (Forts.)

218 Die Größe einer Datei von einem FTP-Server abfragen

Die Größe einer Datei auf einem FTP-Server können Sie über eine Instanz der Klasse `FtpWebRequest` aus dem Namensraum `System.Net` ermitteln. Ein solches Objekt erzeugen Sie über die statische `Create`-Methode der `FtpWebRequest`-Klasse. Da diese (von `WebRequest` geerbte) Methode eine Referenz vom Typ `WebRequest` zurückgibt, müssen Sie die erhaltene Referenz in eine `FtpWebRequest`-Referenz casten.

Erfolgt der Zugriff auf das Internet über einen Proxy-Server, können Sie in die Eigenschaft `Proxy` eine Instanz der Klasse `WebProxy` schreiben, die Sie entsprechend initialisiert haben. In der Eigenschaft `Credentials` können Sie die eventuell notwendigen Benutzerinformationen für den Zugang zum FTP-Server schreiben.

Die Dateigröße fragen Sie dann ab, indem Sie in der Eigenschaft `Method` die FTP-Methode `WebRequestMethods.Ftp.GetFileSize` angeben und die Abfrage über die `GetResponse`-Methode ausführen. Über das zurückgehaltene `FtpWebResponse`-Objekt können Sie in der Eigenschaft `StatusCode` den Status der Abfrage ermitteln. Ist dieser gleich `FtpStatusCode.FileStatus`, war die Abfrage erfolgreich. Nun wird es etwas schwammig, da leider nicht dokumentiert ist, wie die Dateigröße zurückgeliefert wird. Der vom `FtpWebRequest` (laut der Dokumentation) an den FTP-Server gesendete `SIZE`-Befehl ist im FTP-Standard (*www.w3.org/Protocols/rfc959*) leider nicht zu finden. Nach meinen Versuchen ergab die Eigenschaft `ContentLength` die Dateigröße. Ich verlasse mich aber lieber auf den vom Server zurückgelieferten FTP-Status, den Sie über die Eigenschaft `StatusDescription` des `FtpWebResponse`-Objekts auslesen können. Der (im FTP-

Standard dokumentierte) Status 213 (File Status) steht für einen Datei-Status und lieferte in meinen Versuchen einen Status-String in der Form ”213 <Dateigröße>\r\n”. Da ich mich auch beim Zeilenabschluss nicht darauf verlasse, dass dieser (laut dem FTP-Standard) aus ”\r\n” besteht, berücksichtige ich in der Methode `GetFtpFileSize` in Listing 411 alle denkbaren Varianten.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System` und `System.Net` importieren.

```
public static long GetFtpFileSize(string uri, WebProxy proxy,
   NetworkCredential credentials)
{
   // FtpWebRequest erzeugen und die Proxy- und Logindaten übergeben
   FtpWebRequest ftpRequest = (FtpWebRequest)FtpWebRequest.Create(uri);
   if (proxy != null)
   {
      ftpRequest.Proxy = proxy;
   }
   if (credentials != null)
   {
      ftpRequest.Credentials = credentials;
   }

   // Die FTP-Methode festlegen
   ftpRequest.Method = WebRequestMethods.Ftp.GetFileSize;

   // Den Response einlesen
   FtpWebResponse ftpResponse = (FtpWebResponse)ftpRequest.GetResponse();

   // Status überprüfen
   long fileSize = 0;
   if (ftpResponse.StatusCode == FtpStatusCode.FileStatus)
   {
      // Die Dateigröße befindet sich nun in dem Statustext
      // in der Form "213 Dateigröße\r\n", wobei 213 der
      // FTP-Status ist
      string fileSizeString = ftpResponse.StatusDescription.Substring(
         4, ftpResponse.StatusDescription.Length - 4);
      if (fileSizeString.EndsWith("\r\n"))
      {
         fileSizeString = fileSizeString.Substring(0,
           fileSizeString.Length - 2);
      }
      else if (fileSizeString.EndsWith("\r") ||
         fileSizeString.EndsWith("\n"))
```

Listing 411: Methode zur Ermittlung der Dateigröße einer Datei, die auf einem FTP-Server gespeichert ist

```
      {
         fileSizeString = fileSizeString.Substring(0,
           fileSizeString.Length - 1);
      }

      fileSize = Convert.ToInt64(fileSizeString.Trim());
   }

   // Den Response schließen
   ftpResponse.Close();

   // Das Ergebnis zurückgeben
   return fileSize;
}
```

Listing 411: Methode zur Ermittlung der Dateigröße einer Datei, die auf einem FTP-Server gespeichert ist (Forts.)

219 Dateien zu einem Web- oder FTP-Server hochladen

Dateien zu einem Web- oder FTP-Server hochzuladen ist über eine Instanz der `WebClient`-Klasse genauso einfach wie der Download. Dieses Rezept hier ist deswegen mit dem Rezept 215 nahezu identisch, mit dem Unterschied, dass natürlich andere Methoden aufgerufen und andere Ereignisse abgefangen werden. Ich verzichte deswegen auch auf die grundsätzliche Beschreibung der Arbeitsweise der `WebClient`-Klasse. Lesen Sie gegebenenfalls im Rezept 215 nach. Das Rezept 368 beschreibt das ereignisbasierte asynchrone Entwurfsmuster, das die `WebClient`-Klasse implementiert.

Zum Hochladen von Daten oder Dateien über die `WebClient`-Klasse können Sie die in Tabelle 29 beschriebenen Methoden verwenden.

Methode	Bedeutung

`byte[] UploadData(string` ***address***`, byte[]` ***data***`)`
`byte[] UploadData (Uri` ***address***`, byte[]` ***data***`)`
`byte[] UploadData(string` ***address***`, string` ***method***`, byte[]` ***data***`)`
`byte[] UploadData(Uri` ***address***`, string` ***method***`, byte[]` ***data***`)`
	Lädt Daten synchron hoch. Die hochzuladenden Daten übergeben Sie im Argument *data* in Form eines Byte-Arrays. Am Argument *method* können Sie die Methode angeben, die verwendet werden soll. Standardmäßig verwendet `UploadData` für eine HTTP-Verbindung `POST` und für eine FTP-Verbindung `STOR`. Im zurückgegebenen `byte`-Array erhalten Sie die Antwort des Servers.

Tabelle 29: Die Methoden der WebClient-Klasse zum Upload von Daten

Methode	Bedeutung
void UploadDataAsync(URI *address*, byte[] *data*) void UploadDataAsync(URI *address*, string *method*, byte[] *data* [, object *userToken*])	Lädt Daten asynchron hoch. Während des Uploads wird das UploadProgressChanged-Ereignis aufgerufen. Am Ende des Uploads ruft WebClient das UploadDataCompleted-Ereignis auf. In der Eigenschaft Result des Ereignisargument-Objekts erhalten Sie die Antwort des Servers in Form eines Byte-Arrays.
byte[] UploadFile(string *address*, string *fileName*) byte[] UploadFile(Uri *address*, string *fileName*) byte[] UploadFile(string *address*, string *method*, string *fileName*) byte[] UploadFile(Uri *address*, string *method*, string *fileName*)	Lädt eine Datei synchron hoch. Am Argument *method* können Sie die Methode angeben, die verwendet werden soll. Standardmäßig verwendet UploadFile für eine HTTP-Verbindung POST und für eine FTP-Verbindung STOR. Im zurückgegebenen byte-Array erhalten Sie die Antwort des Servers.
void UploadFileAsync(URI *address*, string *fileName*) void UploadFileAsync(URI *address*, string *method*, string *fileName* [, object *userToken*])	Lädt eine Datei asynchron hoch. Während des Uploads wird das UploadProgressChanged-Ereignis aufgerufen. Am Ende des Uploads ruft WebClient das UploadFileCompleted-Ereignis auf. In der Eigenschaft Result des Ereignisargument-Objekts erhalten Sie die Antwort des Servers in Form eines Byte-Arrays.
string UploadString(string *address*, string *data*) string UploadString (Uri *address*, string *data*) string UploadString (string *address*, string *method*, string *data*) string UploadString (Uri *address*, string *method*, string *data*)	Lädt einen String synchron hoch. In dem Rückgabewert erhalten Sie die Antwort des Servers.
void UploadStringAsync (URI *address*, string *data*) void UploadStringAsync (URI *address*, string *method*, string *data* [, object *userToken*])	Lädt einen String asynchron hoch. Während des Uploads wird das UploadProgressChanged-Ereignis aufgerufen. Am Ende des Uploads ruft WebClient das UploadStringCompleted-Ereignis auf. Die Eigenschaft Result des Ereignisargument-Objekts verwaltet die Antwort des Servers.

Tabelle 29: Die Methoden der WebClient-Klasse zum Upload von Daten (Forts.)

Zum synchronen Upload einer Datei rufen Sie die `UploadFile`-Methode auf. Falls der Zugriff auf das Internet über einen Proxy-Server erfolgt, können Sie der `WebClient`-Instanz die Proxy-Informationen in der Eigenschaft `Proxy` übergeben, indem Sie ein neues `WebProxy`-Objekt in dieser Eigenschaft schreiben. Dabei sollten Sie aber beachten, dass das von `WebClient` intern verwendete `FtpWebRequest`- oder `HttpWebRequest`-Objekt per Voreinstellung die im Internet Explorer eingestellten Proxy-Informationen verwendet, falls keine speziellen angegeben sind. Das Setzen des Proxy können Sie sich also unter Umständen sparen. Erfordert der Web- oder FTP-Server eine Authentifizierung, können Sie die dazu notwendigen Informationen über eine neue `NetworkCredential`-Instanz in die Eigenschaft `Credentials` schreiben. Dem Upload einer Datei steht dann nichts mehr im Wege (außer dass der Server die Annahme verweigert ☺).

Zum Kompilieren des folgenden Beispiels, das die Datei *C:\CSS.pdf* zum lokalen FTP-Server hochlädt, müssen Sie die Namensräume `System`, `System.Net` und `System.Windows.Forms` importieren.


```
private void UploadFileSync()
{
   // WebClient erzeugen
   WebClient webClient = new WebClient();

   // Die Proxy-Daten übergeben (falls notwendig)
   string proxyAddress = null;
   int proxyPort = 8080;
   string proxyUserName = null;
   string proxyPassword = null;
   string proxyUserDomain = null;
   if (proxyAddress != null && proxyAddress.Length > 0)
   {
      WebProxy webProxy = new WebProxy(proxyAddress, proxyPort);
      webProxy.Credentials = new NetworkCredential(
         proxyUserName, proxyPassword,
         proxyUserDomain);
      webClient.Proxy = webProxy;
   }

   // Die Authentifizierungsdaten übergeben (falls notwendig)
   string username = null;
   string password = null;
   if (username != null && username.Length > 0)
   {
      webClient.Credentials = new NetworkCredential(
         username, password);
   }

   try
```

Listing 412: Einfacher synchroner Upload einer Datei

```
   {
      // Die Datei herunterladen
      string destinationUri = "ftp://localhost/Css.pdf";
      string sourceFilename = "C:\\Css.pdf";
      webClient.UploadFile(destinationUri, sourceFilename);

      // Ergebnis melden
      MessageBox.Show("Fertig", Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Information);
   }
   catch (WebException ex)
   {
      // Fehler melden
      MessageBox.Show("Fehler beim Upload: " +
         GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
   catch (Exception ex)
   {
      // Fehler melden
      MessageBox.Show(GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
}

/* Liefert alle Nachrichten der übergebenen und der inneren Exceptions */
private static string GetExceptionMessages(Exception ex)
{
   string messages = ex.Message;
   if (ex.InnerException != null)
   {
      messages += Environment.NewLine;
      messages += GetExceptionMessages(ex.InnerException);
   }
   return messages;
}
```

Listing 412: Einfacher synchroner Upload einer Datei (Forts.)

Hinweis

Die Methode `GetExceptionMessages` in Listing 412 habe ich deswegen implementiert, weil die Meldung der von `WebClient` geworfenen `WebException`-Ausnahmen meist nicht besonders aussagekräftig ist und der eigentliche Fehler sich in der inneren Exception versteckt.

Für den asynchronen Upload sollten Sie die `WebClient`-Instanz auf Klassenebene deklarieren, da Sie diese zum Abbrechen des Uploads über die `CancelAsync`-Methode benötigen. Während des Uploads erhalten Sie das Ereignis `UploadProgressChanged`. Das Ende des Uploads wird beim Upload einer Datei über das Ereignis `UploadFileCompleted` gemeldet. Zu diesen Ereignissen sollten Sie im Konstruktor der Klasse Methoden zuweisen.

Das folgende Beispiel basiert auf einem Formular mit Namen `StartForm`. Zum Kompilieren müssen Sie die Namensräume `System`, `System.Windows.Forms` und `System.Net` importieren.

```
/* Referenz auf den WebClient */
private WebClient webClient;

/* Konstruktor des Formulars */
public StartForm()
{
   InitializeComponent();

   // WebClient erzeugen
   this.webClient = new WebClient();

   // Ereignisse zuweisen
   this.webClient.UploadProgressChanged +=
      new UploadProgressChangedEventHandler(
      this.webClient_UploadProgressChanged);
   this.webClient.UploadFileCompleted +=
      new UploadFileCompletedEventHandler(
      this.webClient_UploadFileCompleted);
}
```

Listing 413: Erzeugen und Initialisieren der WebClient-Instanz

Den asynchronen Upload einer Datei starten Sie dann über die `UploadFileAsync`-Methode. Zuvor sollten Sie gegebenenfalls notwendige Proxy-Informationen und Authentifizierungsdaten übergeben:

```
private void UploadAsync()
{
   // Die Proxy-Daten übergeben (falls notwendig)
   string proxyAddress = null;
   int proxyPort = 8080;
   string proxyUserName = null;
   string proxyPassword = null;
   string proxyUserDomain = null;
   if (proxyAddress != null && proxyAddress.Length > 0)
   {
```

Listing 414: Asynchroner Upload einer Datei

```
      WebProxy webProxy = new WebProxy(proxyAddress, proxyPort);
      webProxy.Credentials = new NetworkCredential(
         proxyUserName, proxyPassword,
         proxyUserDomain);
      webClient.Proxy = webProxy;
   }

   // Die Authentifizierungsdaten übergeben (falls notwendig)
   string username = null;
   string password = null;
   if (username != null && username.Length > 0)
   {
      this.webClient.Credentials = new NetworkCredential(
         username, password);
   }

   try
   {
      // Das Hochladen der Datei starten
      Uri destinationUri = new Uri("ftp://localhost/Css.pdf");
      string sourceFilename = "C:\Css.pdf";
      this.webClient.UploadFileAsync(destinationUri, sourceFilename);
   }
   catch (WebException ex)
   {
      // Fehler melden
      MessageBox.Show("Fehler beim Upload: " +
         GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
   catch (Exception ex)
   {
      // Fehler melden
      MessageBox.Show(GetExceptionMessages(ex),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
}
```

Listing 414: Asynchroner Upload einer Datei (Forts.)

Abbrechen können Sie den asynchronen Upload, indem Sie die `CancelAsync`-Methode aufrufen:

```
this.webClient.CancelAsync();
```

Im Ereignis `UploadProgressChanged` wird der Fortschritt des Uploads gemeldet. Die Eigenschaft `BytesSent` des Ereignisargument-Objekts liefert eine Information darüber, wie viele Bytes bis-

her gesendet wurden. Über `TotalBytesToSend` erfahren Sie die Anzahl der insgesamt zu sendenden Bytes, `ProgressPercentage` liefert den prozentualen Fortschritt. Damit können Sie zum Beispiel ein `StatusLabel` und eine `ProgressBar` aktualisieren:

```
private void webClient_UploadProgressChanged(object sender,
   UploadProgressChangedEventArgs e)
{
   long bytesSent = e.BytesSent;
   long totalBytesToSend = e.TotalBytesToSend;
   int progressPercentage = e.ProgressPercentage;
   this.toolStripStatusLabel.Text = bytesSent + " Bytes von " +
      totalBytesToSend + " gesendet ...";
   this.progressBar.Value = progressPercentage;
}
```

Listing 415: Methode für das UploadProgressChanged-Ereignis

In der Methode für das `UploadFileCompleted`-Ereignis sollten Sie (dem ereignisbasierten asynchronen Entwurfsmuster entsprechend) über die Eigenschaft `Cancelled` abfragen, ob der Upload abgebrochen wurde. Falls während des Uploads ein Fehler auftritt, verwaltet die `Error`-Eigenschaft das entsprechende `Exception`-Objekt. Ist `Cancelled` `false` und steht `Error` auf `null`, wurde der Upload erfolgreich beendet;

```
private void webClient_UploadFileCompleted(object sender,
   UploadFileCompletedEventArgs e)
{
   if (e.Cancelled)
   {
      this.toolStripStatusLabel.Text = "Abgebrochen";
   }
   else if (e.Error != null)
   {
      this.toolStripStatusLabel.Text = "Fehler";
      MessageBox.Show(GetExceptionMessages(e.Error),
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
   }
   else
   {
      this.toolStripStatusLabel.Text = "Fertig";
   }

   this.progressBar.Value = 0;
}
```

Listing 416: Methode für das UploadFileCompleted-Ereignis

220 FTP-Verzeichnisse auslesen

Zum Auslesen von FTP-Verzeichnissen verwenden Sie eine Instanz der `FtpWebRequest`-Klasse aus dem Namensraum `System.Net`. In der Eigenschaft `Method` geben Sie die zu verwendende FTP-Methode an. Zum Auslesen eines Verzeichnisses können Sie die Methoden *NLST* oder *LIST* verwenden, die Sie auch über die Eigenschaften `ListDirectory` und `ListDirectoryDetails` der `WebRequestMethods.Ftp`-Klasse angeben können. `ListDirectory` (NLST) gibt nur die Namen der Dateien beziehungsweise Ordner aus dem angegebenen FTP-Verzeichnis zurück, `ListDirectoryDetails` (LIST) liefert neben den Namen zusätzliche Informationen, wie zum Beispiel die Dateigröße. Dummerweise ist das Format, in dem die erweiterten Informationen zurückgegeben werden, nicht festgelegt. Unix- oder Linux-Server liefern Detailinformationen in der Regel im UNIX-Format zurück. Für eine Datei wird z. B. der folgende String geliefert:

```
-rwxrwxrwx   1 owner    group          609277 Jan 12 14:09 Css.pdf
```

Ein Ordner führt zu dem folgenden String:

```
drwxrwxrwx   1 owner    group               0 Jan 13 12:05 CSharp
```

Windows-Server liefern Detailinformationen meist im DOS-Format. Für eine Datei wird z. B. der folgende String zurückgegeben:

```
01-12-06  02:09PM               609277 Css.pdf
```

Für einen Ordner liefert das DOS-Format die folgende Angabe:

```
01-13-06  12:05PM       <DIR>          CSharp
```

Andere Betriebssysteme verwenden unter Umständen ein anderes Format, aber das UNIX- und das DOS-Format sind sehr weit verbreitet und damit Quasi-Standard. Ein Programm sollte in meinen Augen auf jeden Fall in der Lage sein, die Detailinformationen der Verzeichniseinträge von einem FTP-Server abzufragen und anzuzeigen. Die bloße Anzeige des zurückgelieferten Strings ist einem Benutzer wohl kaum zuzumuten. Deswegen habe ich mich entschlossen, die Standard-Formate zu unterstützen und die Detailinformationen auszulesen (was nicht allzu einfach war). Dieses Rezept ist deswegen auch etwas länger, aber ich denke, dass Sie (und ich) etwas davon haben ☺.

Die Methode `ListFtpDirectory` in Listing 421 liest ein FTP-Verzeichnis ein. Neben dem String mit dem URI des FTP-Verzeichnisses kann dieser Methode ein Objekt übergeben werden, das die `ICredentials`-Schnittstelle implementiert (z. B. ein `NetworkCredential`-Objekt). Die in diesem Objekt verwalteten Benutzerdaten werden für eine Authentifizierung gegenüber dem FTP-Server verwendet. Soll die anonyme Authentifizierung erfolgen, übergeben Sie an diesem Argument `null`. Das Argument `proxy` dient der Angabe von Proxy-Informationen für den Fall, dass der Zugriff auf das Internet über einen Proxy-Server erfolgt. Das letzte Argument gibt an, ob Details ausgegeben werden sollen. Die Methode gibt eine `ReadOnlyCollection` (aus dem Namensraum `System.Collections.ObjectModel`) mit Instanzen der eigenen Klasse `FtpDirectoryItem` zurück, die die einzelnen Einträge darstellen.

Diese Klasse bildet dann auch die Basis für das Zerlegen der Verzeichniseinträge.

Zum Kompilieren des Quellcodes dieses Rezepts müssen Sie die Namensräume `System`, `System.Net`, `System.IO`, `System.Collections.Generic`, `System.Collections.ObjectModel`, `System.Text`, `System.Globalization` und `System.Text.RegularExpressions` einbinden.

Zunächst habe ich aber die Aufzählung `FtpDirectoryItemType` implementiert, die später als Typ der Eigenschaft `Type` verwendet wird:

```
public enum FtpDirectoryItemType
{
   /* Verzeichnis */
   Directory,

   /* Datei */
   File,

   /* (UNIX-)Link */
   Link,

   /* Unbekannter Typ */
   Unknown
}
```

Listing 417: Aufzählung für den Typ eines FTP-Verzeichniseintrags

In der Klasse `FtpDirectoryItem` habe ich zunächst schreibgeschützte Eigenschaften für die Detailinformationen der Verzeichniseinträge implementiert:

```
public class FtpDirectoryItem
{
   private string name;
   /* Der Name des Verzeichniseintrags */
   public string Name
   {
      get { return this.name; }
   }

   private FtpDirectoryItemType type;
   /* Der Typ des Verzeichniseintrags */
   public FtpDirectoryItemType Type
   {
      get { return this.type; }
   }

   private DateTime date;
   /* Das Datum des Verzeichniseintrags */
   public DateTime Date
   {
      get { return this.date; }
   }
```

Listing 418: Die Eigenschaften der Klasse FtpDirectoryItem

```
private long size;
/* Die Größe des Verzeichniseintrags  */
public long Size
{
   get { return this.size; }
}
private string rawItem;
/* Der Verzeichniseintrag in der rohen, vom Server gelieferten Form */
public string RawItem
{
   get { return this.rawItem; }
}
```

Listing 418: Die Eigenschaften der Klasse FtpDirectoryItem (Forts.)

Im UNIX-Format werden Monatsnamen üblicherweise in der englischen abgekürzten Schreibweise angegeben. Zum Konvertieren eines solchen Monats in den entsprechenden `int`-Wert habe ich die folgende Methode implementiert:

```
private int ConvertEnglishAbbreviatedMonthNameToInt(string monthName)
{
   switch (monthName)
   {
      case "Jan":
         return 1;
      case "Feb":
         return 2;
      case "Mar":
         return 3;
      case "Apr":
         return 4;
      case "May":
         return 5;
      case "Jun":
         return 6;
      case "Jul":
         return 7;
      case "Aug":
         return 8;
      case "Sep":
         return 9;
      case "Oct":
         return 10;
      case "Nov":
         return 11;
```

Listing 419: Methode zum Konvertieren eines englischen, abgekürzten Monatsnamens in den entsprechenden int-Wert

```
            case "Dec":
               return 12;
            default:
               return 0;
         }
      }
```

Listing 419: Methode zum Konvertieren eines englischen, abgekürzten Monatsnamens in den entsprechenden int-Wert (Forts.)

Die eigentliche Arbeit, das Zerlegen eines Verzeichniseintrags, übernimmt der Konstruktor der `FtpDirectoryItem`-Klasse. Wenn Details angezeigt werden sollen (was über das `listDetails`-Argument der `ListFtpDirectory`-Methode gesteuert wird), beginnt die eigentliche (und komplizierte) Arbeit.

In diesem Fall ermittelt der Konstruktor zunächst das Format der Einträge. DOS-Einträge werden daran erkannt, dass diese größer sind als 38 Zeichen und mit einem US-amerikanischen Datum in den ersten 17 Zeichen beginnen. Um dies zu erkennen ruft der Konstruktor die `TryParse`-Methode der `DateTime`-Klasse auf und übergibt als Format-Provider ein `CultureInfo`-Objekt, das die US-amerikanische Kultur repräsentiert (und das aus Performancegründen von `ListFtpDirectory` erzeugt und für jeden Eintrag übergeben wird).

Bei DOS-Einträgen ist das Zerlegen recht einfach. Ich denke, die Kommentare im Quelltext sprechen für sich.

Bei UNIX-Einträgen wird es etwas komplizierter. Die Basis meines Algorithmus fand ich in dem Artikel »Appendix: FTP LIST Specification« an der Adresse *www.interarchy.com/documentation/6/pages/appendix-ftp-list.html*

Zu ermitteln, ob es sich um eine Datei, ein Verzeichnis oder einen UNIX-Link handelt (der anders als bei Windows richtig im Betriebssystem verankert ist), ist noch einfach: Der Eintrag beginnt entweder mit einem Bindestrich (Datei), einem *d* (Verzeichnis) oder einem *l* (Link). Für andere Fälle habe ich den Typ `FtpDirectoryItemType.Unknown` vorgesehen.

Danach sucht der Konstruktor den Datumseintrag. Dazu habe ich zwei reguläre Ausdrücke verwendet, die recht komplex geworden sind. Der erste reguläre Ausdruck sucht nach einem Datum im Format *<Abgekürzter Monatsname> <Tag> <Jahr>*. Dies ist das Format, in dem im UNIX-Format Datumswerte angegeben sind, wenn das Jahr nicht das zum Monat und zum Tag nächste passende, in der Vergangenheit liegende Jahr ist. In anderen Fällen verwenden diese Server das Format *<Abgekürzter Monatsname> <Tag> <Stunde>:<Minute>*. Bei diesem Format ist davon auszugehen, dass das Jahr das nächste, zum Rest des Datums passende, in der Vergangenheit liegende Jahr ist. Ist das aktuelle Datum z. B. der 15.2.2006 und ein UNIX-Server gibt "JAN 15 10:10" zurück, wäre das nächste passende Jahr 2006, weil das Datum 15.1.2006 10:10 in der Vergangenheit liegt. Würde der UNIX-Server allerdings "MAR 15 10:10" zurückgeben, wäre es das Jahr 2005, weil der 15.3.2006 (bezogen auf den 15.2.2006) nicht in der Vergangenheit liegen würde. Die regulären Ausdrücke berücksichtigen beide Formate und erlauben zwischen den Datumsteilen beliebig viele Leerzeichen.

In den regulären Ausdrücken habe ich für die wichtigen Bestandteile Gruppen eingefügt und diese benannt. So konnte ich später relativ einfach auf die einzelnen Teile zugreifen und diese

in ein Datum konvertieren. Bei Auswerten des Ergebnisses des jeweiligen regulären Ausdrucks bin ich davon ausgegangen, dass das Datum diejenige Fundstelle ist, die an erster Stelle steht. Schließlich kann auch vorkommen, dass im Namen des Eintrags ebenfalls ein Datum angegeben ist.

Nachdem der Datumseintrag ermittelt wurde, sucht das Programm lediglich noch nach dem Größen-Eintrag, der durch Leerzeichen getrennt links vom Datum angegeben ist, und ermittelt den Namen des Eintrags, der rechts vom Datum steht.

Für den Fall, dass keine Details angezeigt werden sollen, wird unten im `else`-Block lediglich der Name des Eintrags gespeichert.

```
internal FtpDirectoryItem(string rawItem, bool isDetails,
   CultureInfo usCulture)
{
   // Den rohen Eintrag ablegen
   this.rawItem = rawItem;

   if (isDetails)
   {
      // Zerlegen des Eintrags
      // Zunächst überprüfen, ob es ein DOS- oder UNIX-Eintrag ist
      bool isDos = false;
      if (rawItem.Length >= 39)
      {
         string testValue = rawItem.Substring(0, 17);
         if (DateTime.TryParse(testValue, usCulture, DateTimeStyles.None,
            out this.date))
         {
            // Der Eintrag beginnt mit einem (US-amerikanischen) Datum:
            // Es ist (mit hoher Wahrscheinlichkeit) ein DOS-Eintrag
            isDos = true;
         }
      }

      if (isDos)
      {
         // DOS-Eintrag
         // Überprüfen, ob an Position 24 <DIR> steht
         if (rawItem.Substring(24, 5).ToLower() == "<dir>")
         {
            // Es handelt sich um ein Verzeichnis
            this.type = FtpDirectoryItemType.Directory;
            this.size = 0;
         }
         else
         {
```

Listing 420: Konstruktor der FtpDirectoryItem-Klasse und abschließende Klammer

```
            // Es handelt sich nicht um ein Verzeichnis
            this.type = FtpDirectoryItemType.File;

            // Die Größe auslesen
            long.TryParse(rawItem.Substring(29, 9), out this.size);
         }

         // Den Namen auslesen
         if (rawItem.Length > 39)
         {
            this.name = rawItem.Substring(39, rawItem.Length - 39).Trim();
         }
      }
      else
      {
         // UNIX-Eintrag
         // Überprüfen, ob der Eintrag mit d, - oder l beginnt
         switch (rawItem[0])
         {
            case 'd':
               this.type = FtpDirectoryItemType.Directory;
               break;

            case '-':
               this.type = FtpDirectoryItemType.File;
               break;

            case 'l':
               this.type = FtpDirectoryItemType.Link;
               break;

            default:
               this.type = FtpDirectoryItemType.Unknown;
               break;
         }

         // Den Datumseintrag suchen
         int year = 0;
         int month = 0;
         int day = 0;
         int hour = 0;
         int minute = 0;
         // Erst im Format MMM DD YYYY oder MMM D YYYY
         string pattern = "(?<month>(Jan)|(Feb)|(Mar)|(Apr)|(May)" +
            "|(Jun)|(Jul)|(Aug)|(Sep)|(Oct)|(Nov)|(Dec)){1}" +
            "( ){1,}(?<day>[0-9]([0-9]){0,1})( ){1,}" +
```

Listing 420: Konstruktor der FtpDirectoryItem-Klasse und abschließende Klammer (Forts.)

```
            "(?<year>[0-9][0-9][0-9][0-9])";
        MatchCollection matches = Regex.Matches(rawItem, pattern);
        if (matches.Count > 0)
        {
            // Die erste Fundstelle enthält das Datum
            int.TryParse(matches[0].Groups["year"].Value, out year);
            month = this.ConvertEnglishAbbreviatedMonthNameToInt(
               matches[0].Groups["month"].Value);
            int.TryParse(matches[0].Groups["day"].Value, out day);
            if (year > 0 && month > 0 && day > 0)
            {
               this.date = new DateTime(year, month, day);
            }
        }
        else
        {
            // Dann im Format MMM DD HH:MM oder MMM D HH:MM
            pattern = "(?<month>(Jan)|(Feb)|(Mar)|(Apr)|(May)" +
               "|(Jun)|(Jul)|(Aug)|(Sep)|(Oct)|(Nov)|(Dec))" +
               "( ){1,}(?<day>[0-9]([0-9]){0,1})( ){1,}" +
               "(?<hour>[0-9][0-9])\\:(?<minute>[0-9][0-9])";
            matches = Regex.Matches(rawItem, pattern);

            // Die erste Fundstelle ist das Datum, das Jahr muss aber
            // ermittelt werden
            // Das Jahr ermittelt sich aus dem nächstpassenden Jahr
            // in der Vergangenheit
            month = this.ConvertEnglishAbbreviatedMonthNameToInt(
               matches[0].Groups["month"].Value);
            int.TryParse(matches[0].Groups["day"].Value, out day);
            int.TryParse(matches[0].Groups["hour"].Value, out hour);
            int.TryParse(matches[0].Groups["minute"].Value, out minute);
            if (month > 0 && day > 0)
            {
               this.date = new DateTime(DateTime.Now.Year, month,
                  day, hour, minute, 0);
               if (this.Date > DateTime.Now)
               {
                  this.date = new DateTime(DateTime.Now.Year - 1, month,
                     day, hour, minute, 0);
               }
            }
        }

        if (matches.Count > 0)
        {
```

Listing 420: Konstruktor der FtpDirectoryItem-Klasse und abschließende Klammer (Forts.)

```
                // Direkt vor dem gefundenen Datum liegt die Dateigröße
                // Nach dem nächsten Leerzeichen nach links suchen
                for (int i = matches[0].Index - 2; i >= 0; i--)
                {
                   if (rawItem[i] == ' ')
                   {
                      // Leerzeichen gefunden: Die Größe auslesen
                      string sizeString = rawItem.Substring(i + 1,
                         matches[0].Index - i - 1).Trim();
                      long.TryParse(sizeString, out this.size);
                      break;
                   }
                }

                // Hinter dem Datum liegt der Name
                int namePos = matches[0].Index + matches[0].Length + 1;
                this.name = rawItem.Substring(namePos,
                   rawItem.Length - namePos).Trim();
             }
          }
       }
       else
       {
          // Keine Details, also nur der Name
          this.name = rawItem;
          this.type = FtpDirectoryItemType.Unknown;
       }
    }
}
```

Listing 420: Konstruktor der FtpDirectoryItem-Klasse und abschließende Klammer (Forts.)

`ListFtpDirectory` ist schließlich die Methode, die das FTP-Verzeichnis ausliest. Nachdem das `FtpWebRequest`-Objekt erzeugt und initialisiert wurde, liest `ListFtpDirectory` über die `GetResponse`-Methode die Antwort des FTP-Servers ein und ermittelt den Antwort-Stream. Dieser wird in einen `StreamReader` gelesen, wobei ich die im Internet übliche Codierung ISO-8859-1 übergeben habe. An dieser Stelle war ich mir nicht sicher, ob ISO-8859-1 in allen Fällen funktioniert. Mit UTF8 gab es aber Probleme mit Dateien, deren Name Umlaute beinhaltete. ISO-8859-1 funktionierte in meinen Tests einwandfrei.

`ListFtpDirectory` liest dann die einzelnen Zeilen in eine Instanz der Klasse `List<FtpDirectoryItem>`, die schließlich als Basis für die zurückgegebene `ReadOnlyCollection` verwendet wird.

```
public static ReadOnlyCollection<FtpDirectoryItem> ListFtpDirectory(
   string uriString, ICredentials credentials, IWebProxy proxy,
   bool listDetails)
```

Listing 421: Methode zum Auslesen eines FTP-Verzeichnisses

```
{
   // List-Instanz für das Ergebnis
   List<FtpDirectoryItem> directoryList = new List<FtpDirectoryItem>();

   // FtpWebRequest erzeugen und die FTP-Methode festlegen
   FtpWebRequest request = (FtpWebRequest)FtpWebRequest.Create(uriString);
   if (listDetails)
   {
      request.Method = WebRequestMethods.Ftp.ListDirectoryDetails;
   }
   else
   {
      request.Method = WebRequestMethods.Ftp.ListDirectory;
   }

   // Credentials und Proxy übergeben
   if (credentials != null)
   {
      request.Credentials = credentials;
   }
   if (proxy != null)
   {
      request.Proxy = proxy;
   }

   Stream responseStream = null;
   StreamReader sr = null;
   try
   {
      // Die Antwort holen
      FtpWebResponse response = (FtpWebResponse)request.GetResponse();

      // Den Antwort-Stream holen und in einen StreamReader lesen
      responseStream = response.GetResponseStream();
      sr = new StreamReader(responseStream,
         Encoding.GetEncoding("iso-8859-1"));

      // CultureInfo-Objekt für die US-amerikanische Kultur erzeugen
      CultureInfo usCulture = CultureInfo.CreateSpecificCulture("en-US");

      // Das Ergebnis lesen
      string row;
      while ((row = sr.ReadLine()) != null)
      {
         directoryList.Add(new FtpDirectoryItem(row, listDetails,
            usCulture));
```

Listing 421: Methode zum Auslesen eines FTP-Verzeichnisses (Forts.)

```
        }
    }
    finally
    {
        // StreamReader (und damit implizit den Antwort-Stream schließen)
        try
        {
            sr.Close();
        }
        catch { }
    }

    // Das Ergebnis zurückgeben
    return new ReadOnlyCollection<FtpDirectoryItem>(directoryList);
}
```

Listing 421: Methode zum Auslesen eines FTP-Verzeichnisses (Forts.)

Abbildung 123 zeigt die Beispielanwendung zu diesem Rezept, die gerade das Verzeichnis eines FTP-Servers in ein `DataGridView`-Steuerelement gelesen hat.

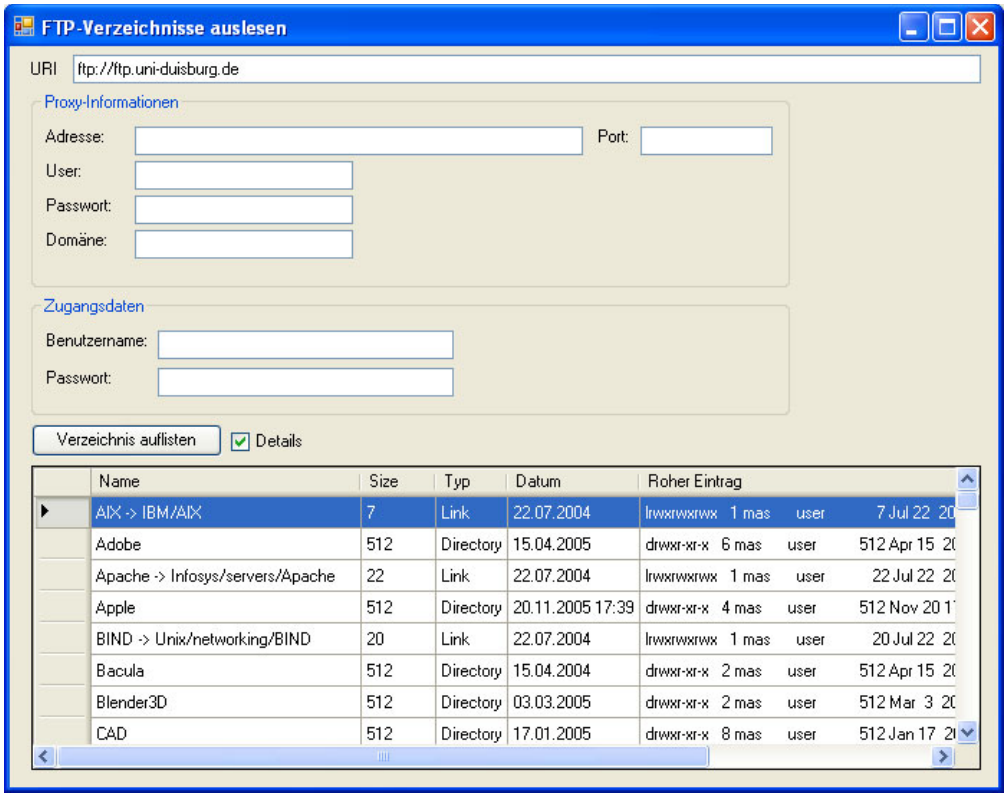


Abbildung 123: Beispielanwendung zum Auslesen von FTP-Verzeichnissen

Hinweis

Das Auslesen von FTP-Verzeichnissen funktioniert natürlich nur dann, wenn der Server den DOS- oder UNIX-Standard verwendet. Einige Server sollen angeblich im UNIX-Format die Monatsnamen in französischer Form ausgeben. Ich konnte dies leider nicht nachvollziehen. Für die Fälle, bei denen das Format nicht erkannt wird, bleibt die `Name`-Eigenschaft der `FtpDirectoryItem`-Objekte (hoffentlich in allen Fällen) leer.

Falls Sie dieses Rezept um die Unterstützung weiterer Formate erweitern oder verbessern, würde ich mich über eine E-Mail freuen und Ihre Erweiterung natürlich im Erratum veröffentlichen.

221 Dateien und Ordner auf einem FTP-Server löschen

Wenn Sie auf einem FTP-Server Dateien oder Ordner löschen wollen, benötigen Sie eine Instanz der `FtpWebRequest`-Klasse, deren Eigenschaft `Method` Sie zum Löschen einer Datei auf `WebRequestMethods.Ftp.DeleteFile` und zum Löschen eines Ordners auf `WebRequestMethods.Ftp.RemoveDirectory` festlegen. Nachdem Sie gegebenenfalls notwendige Authentifizierungs- und Proxy-Informationen übergeben haben, führen Sie die Anforderung über die `GetResponse`-Methode aus. Falls die Datei oder der Ordner nicht gelöscht werden kann, erhalten Sie eine `WebException` mit einer entsprechenden Fehlermeldung. Dateien und Ordner können natürlich nur dann gelöscht werden, wenn der FTP-Benutzer, mit dem das Programm sich bei dem Server angemeldet hat, das Lösch-Recht besitzt. Ordner können auf vielen FTP-Servern nur dann gelöscht werden, wenn diese leer sind.

Die Methode `DeleteFtpFileOrFolder` in Listing 422 übernimmt das Löschen einer Datei oder eines Ordners. Am ersten Argument geben Sie den URI der Datei bzw. das Ordners und am zweiten und dritten Argument gegebenenfalls notwendige Authentifizierungs- und Proxy-Informationen an. Das letzte Argument legt fest, ob es sich um eine Datei handelt. Geben Sie hier `false` an, handelt es sich um einen Ordner.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System.Net` importieren.

```
public static void DeleteFtpFileOrFolder(string uriString,
   ICredentials credentials, IWebProxy proxy, bool isFile)
{
   // FtpWebRequest erzeugen und die FTP-Methode festlegen
   FtpWebRequest request = (FtpWebRequest)FtpWebRequest.Create(uriString);
   if (isFile)
   {
      request.Method = WebRequestMethods.Ftp.DeleteFile;
   }
   else
   {
      request.Method = WebRequestMethods.Ftp.RemoveDirectory;
   }
```

Listing 422: Methode zum Löschen einer Datei oder eines (leeren) Ordners

```
   // Credentials und Proxy übergeben
   if (credentials != null)
   {
      request.Credentials = credentials;
   }
   if (proxy != null)
   {
      request.Proxy = proxy;
   }

   // Den Request ausführen
   request.GetResponse();
}
```

Listing 422: Methode zum Löschen einer Datei oder eines (leeren) Ordners (Forts.)

Hinweis

Das Löschen können Sie idealerweise auf dem lokalen Windows-FTP-Server ausprobieren. Beachten Sie dabei, dass in den Eigenschaften des FTP-Servers das Schreiben in das Basisverzeichnis erlaubt sein muss. Außerdem muss der angemeldete Benutzer auf der Dateisystem-Ebene ebenfalls Schreibrechte auf der zu löschenden Datei bzw. auf dem zu löschenden Ordner besitzen. Ist der anonyme Zugriff erlaubt, müssen Sie diese Rechte dem Benutzer *IUSR_<Computername>* vergeben.

Hinweis

Was dieses Rezept nicht beinhaltet, ist das Löschen von Ordnern, die Dateien oder Unterordner besitzen, auf FTP-Servern, die das Löschen solcher Ordner nicht erlauben. Das Löschen solcher Ordner ist leider nicht allzu einfach, da dazu alle Dateien und Ordner ermittelt werden und die Unterordner rekursiv behandelt werden müssen. Leider hat Microsoft keine Möglichkeit geschaffen, einen Ordner trotz enthaltener Dateien und Unterordner zu löschen.

222 Ordner auf einem FTP-Server erstellen

Zum Erstellen eines Ordners auf einem FTP-Server erzeugen Sie eine Instanz der `FtpWebRequest`-Klasse und legen die Methode auf `WebRequestMethods.Ftp.MakeDirectory` fest. Für den Fall, dass der FTP-Server eine Authentifizierung erfordert, schreiben Sie in die `Credentials`-Eigenschaft ein `NetworkCredential`-Objekt mit den entsprechenden Informationen. Erfolgt der Zugriff auf das Internet über einen Proxy-Server, schreiben Sie zusätzlich in die Eigenschaft `Proxy` eine Instanz der `WebProxy`-Klasse, die Sie entsprechend initialisiert haben. Danach können Sie die Anforderung über die `GetResponse`-Methode ausführen. Existiert der Ordner bereits, oder besitzt der am FTP-Server angemeldete Benutzer keine Schreibrechte, resultiert die Anforderung in einer `WebException` mit einer entsprechenden Meldung.

Die Methode `CreateFtpFolder` in Listing 423 legt einen Ordner auf einem FTP-Server an. Am ersten Argument übergeben Sie den URI des anzulegenden Ordners. Am zweiten Argument können Sie Authentifizierungs-Informationen übergeben. Soll die Authentifizierung anonym erfolgen, übergeben Sie hier `null`. Dabei ist natürlich unwahrscheinlich, dass der anonyme

Benutzer Schreibrechte besitzt. Am letzten Argument können Sie gegebenenfalls notwendige Proxy-Informationen übergeben.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System.Net` importieren.

```
public static void CreateFtpFolder(string uriString,
   ICredentials credentials, IWebProxy proxy)
{
   // FtpWebRequest erzeugen und die FTP-Methode festlegen
   FtpWebRequest request = (FtpWebRequest)FtpWebRequest.Create(uriString);
   request.Method = WebRequestMethods.Ftp.MakeDirectory;

   // Credentials und Proxy übergeben
   if (credentials != null)
   {
      request.Credentials = credentials;
   }
   if (proxy != null)
   {
      request.Proxy = proxy;
   }

   // Request ausführen
   request.GetResponse();
}
```

Listing 423: Methode zum Erzeugen eines Ordners auf einem FTP-Server

Bei der Anwendung der Methode sollten Sie natürlich Ausnahmen abfangen:

```
// NetworkCredential-Objekt für die Authentifizierung erzeugen
string userName = "zaphod";
string password = "galaxy";
NetworkCredential credentials = new NetworkCredential(userName, password);

try
{
   string uriString = "ftp://localhost/Demo";
   FtpUtils.CreateFtpFolder(uriString, credentials, null);
   Console.WriteLine("Ordner erfolgreich erzeugt");
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 424: Beispielhafte Anwendung der CreateFtpFolder-Methode in einer Konsolenanwendung

Windows.Forms

223 Formulare ohne Titelleiste

Eine in Newsgroups häufiger auftauchende Frage ist, wie ein Formular ohne Titelleiste erstellt wird. Eine mögliche Antwort wäre, einfach die Eigenschaft `FormBorderStyle` auf `None` zu setzen. Dann besitzt das Formular aber keinen Rahmen und sieht etwas unschön aus. Die Lösung ist ganz einfach: Setzen Sie die Eigenschaft `ControlBox` auf `false` und löschen Sie den Inhalt der Eigenschaft `Text`. Dann besitzt das Formular den in `FormBorderStyle` definierten Rahmen, aber keine Titelleiste.



Abbildung 124: Formular mit einem veränderbaren Rahmen, ControlBox = false und keinem Titel

224 (Rahmenlose) Formulare über den Clientbereich verschiebbar machen

Ein Formular mit `FormBorderStyle = FormBorderStyle.None`, ohne Titelleiste (siehe Rezept 223) oder mit einer besonderen Form (siehe Rezept 229) kann vom Anwender nicht verschoben werden, da die Titelleiste fehlt. Dieses Problem können Sie aber lösen, indem Sie dem Anwender ermöglichen, das Formular über den Clientbereich zu verschieben. Diese Technik können Sie natürlich auch bei normalen Formularen einsetzen.

Die eleganteste Lösung dazu ist, die Methode `WndProc` des Formulars zu überschreiben. Diese Methode wird von Windows immer dann aufgerufen, wenn eine Windows-Nachricht an das Fenster gesendet wurde. Das ist z. B. dann der Fall, wenn der Anwender die Maus über dem Formular bewegt oder mit der Maus auf dieses klickt.

Die Nachricht können Sie über das By Reference übergebene Argument `m` lesen und auch überschreiben. In der Eigenschaft `Msg` des damit referenzierten `Message`-Objekts wird die Art der Nachricht in Form eines `int`-Werts verwaltet. Die dazu verwendeten Konstanten sind größtenteils in der Headerdatei *winuser.h* deklariert. Die Konstante `WM_NCHITTEST` steht für die Nachricht, dass der Anwender die Maus auf dem Fenster bewegt oder mit der Maus darauf geklickt hat.

Listing 425 zeigt, wie Sie die `WndProc`-Methode so überschreiben, dass das Bewegen des Formulars über den Clientbereich ermöglicht wird. Dabei wird die Nachricht `WM_NCHITTEST` abgefangen und überprüft, ob der Mauscursor sich im Clientbereich des Formulars befindet. Ist dies der Fall, simuliert `WndProc`, dass der Mauscursor auf der Titelleiste bewegt wird, indem die Konstante `HTCAPTION` in das Ergebnis der Nachricht geschrieben und die Methode beendet wird. Am Schluss wird die geerbte `WndProc`-Methode aufgerufen, damit alle anderen Nachrichten korrekt vom Formular verarbeitet werden.

```
protected override void WndProc(ref Message m)
{
   const int WM_NCHITTEST = 0x0084;
   const int HTCAPTION = 2;

   // Abfangen der Nachricht WM_NCHITTEST
   if (m.Msg == WM_NCHITTEST)
   {
      if (this.ClientRectangle.Contains(this.PointToClient(Cursor.Position)))
      {
         // Wenn der Cursor sich im Clientbereich des Formulars befindet:
         // Simulieren, dass der Cursor sich auf der Titelleiste befindet
         m.Result = (IntPtr)HTCAPTION;
         return;
      }
   }

   base.WndProc(ref m);
}
```

Listing 425: Abfangen und Abändern der Nachricht WM_NCHITTEST zum Simulieren der Mauscursorbewegung auf der Titelleiste

Ein Nachteil dieser Technik ist, dass Sie Mausereignisse im Formular nicht mehr über die Formularereignisse auswerten können, da `WndProc` die Mausnachrichten abfängt und nicht an die geerbte `WndProc`-Methode weitergibt. Mausereignisse der Steuerelemente werden allerdings weiter behandelt, da diese eine eigene `WndProc`-Methode besitzen.

Eine andere Lösung des Verschiebe-Problems setzt die API-Funktion `ReleaseCapture` ein, die die Maus freigibt, und die Funktion `SendMessage`, die eine Nachricht an ein Fenster (oder an Windows) sendet. Über die Nachricht `WM_NCLBUTTONDOWN` mit dem Argument `HTCAPTION` können Sie die Betätigung einer Maustaste auf der Titelleiste simulieren.

Dazu benötigen Sie zunächst den Import des Namensraums `System.Runtime.InteropServices` und die Deklaration der verwendeten API-Funktionen und -Konstanten:

```
[DllImport("User32.dll")]
private static extern int SendMessage(IntPtr handle, int msg, int wparam,
   int lparam);

[DllImport("User32.dll")]
public static extern int ReleaseCapture();

private const int WM_NCLBUTTONDOWN = 0x00A1;
private const int HTCAPTION = 2;
```

Listing 426: Deklaration der für die Maus-Simulation benötigten Konstanten

Im MouseDown-Ereignis des Formulars überprüfen Sie, ob die linke Maustaste betätigt wurde. Ist dies der Fall, rufen Sie ReleaseCapture auf um die Maus vom Fenster freizugeben. Über SendMessage senden Sie schließlich die Nachricht WM_NCLBUTTONDOWN mit dem Parameter HTCAPTION:

```
private void WndProcDemoForm_MouseDown(object sender,
    System.Windows.Forms.MouseEventArgs e)
{
   if (e.Button == MouseButtons.Left)
   {
      // Wenn die linke Maustaste betätigt wurde: Maus freigeben
      // und simulieren, dass der Benutzer auf der Titelleiste
      // geklickt hat
      ReleaseCapture();
      SendMessage(Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0);
   }
}
```

Listing 427: Simulieren eines Klicks auf die Titelleiste eines Formulars

Diese Technik ist zwar etwas komplexer, besitzt aber den Vorteil, dass Sie Mausereignisse im Formular ganz normal auswerten können.

225 Unbewegbare Formulare mit Titelleiste

Wenn Sie Formulare unbewegbar machen wollen, können Sie den Formular-Rahmen einfach auf FormBorderStyle.None setzen. Soll das Formular aber einen Titel besitzen, müssen Sie eine andere Technik einsetzen. Sie können dazu im Systemmenü einfach den Befehl zum Verschieben entfernen. Fehlt dieser Befehl, kann der Anwender das Formular auch nicht mehr über die Titelleiste verschieben.

Dazu verwenden Sie die API-Funktion DeleteMenu, der Sie am ersten Argument den Handle des Systemmenüs, am zweiten die Konstante SC_MOVE (für das Verschieben-Menü) und am dritten Argument (an dem Flags übergeben werden) die Konstante MF_BYCOMMAND übergeben (um zu definieren, dass das zweite Argument nicht als Index, sondern als Menü-Id interpretiert werden soll). Den Handle des Systemmenüs erhalten Sie über die API-Funktion GetSystemMenu. Am ersten Argument übergeben Sie den Handle des Formulars, am zweiten bestimmen Sie mit 0 (was dem FALSE des Windows-APIs entspricht), dass eine veränderbare Kopie des Systemmenüs erzeugt werden soll. Das Ganze programmieren Sie entweder im Konstruktor des Formulars oder im Load-Ereignis.

Zum Kompilieren dieses Programms müssen Sie die Namensräume System und System.Runtime.InteropServices importieren.

```
[DllImport("user32.dll")]
private static extern IntPtr GetSystemMenu(IntPtr hWnd, int bRevert);

[DllImport("user32.dll")]
private static extern int DeleteMenu(IntPtr hMenu, int uPosition, int uFlags);

private const int SC_MOVE = 0xF010;
private const int MF_BYCOMMAND = 0;

private void StartForm_Load(object sender, System.EventArgs e)
{
   // Handle des Systemmenüs ermitteln und den Verschieben-Befehl löschen
   IntPtr sysMenuHandle = GetSystemMenu(this.Handle, 0);
   DeleteMenu(sysMenuHandle, SC_MOVE, MF_BYCOMMAND);
}
```

Listing 428: Unbewegbar-Machen eines Formulars

226 Andockende Formulare

Formulare, die am inneren Rand eines Parent-Formulars bzw. aneinander andocken, wie in Abbildung 125 gezeigt, können Sie recht einfach implementieren.

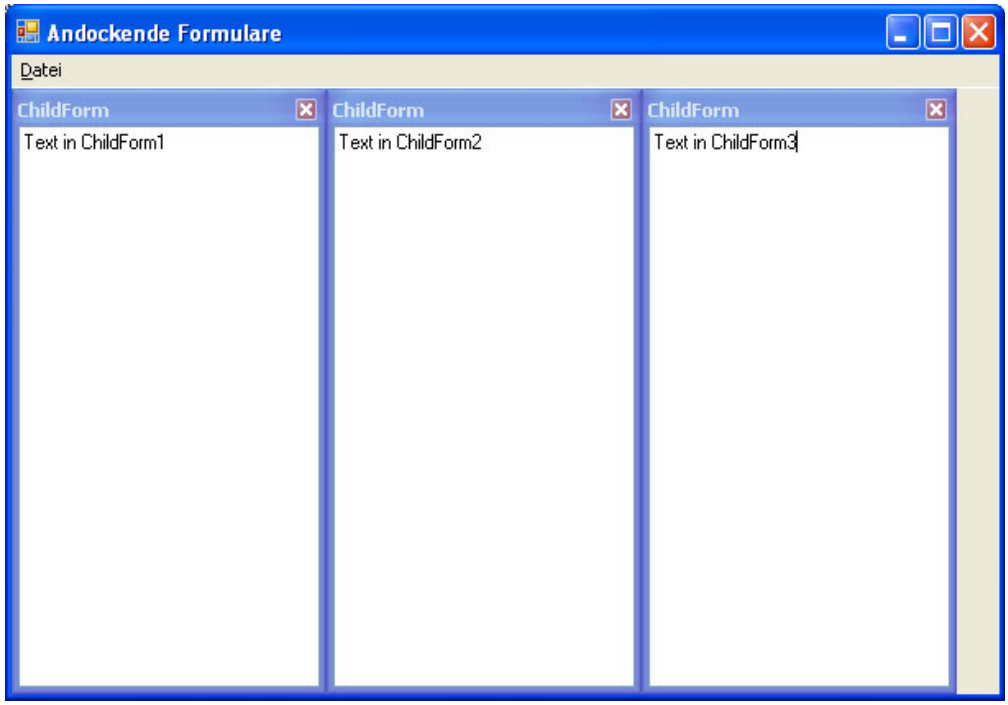


Abbildung 125: Andockende Formulare

Der Trick ist, dass Sie das neue Formular der `Controls`-Auflistung des Parent-Formulars zuweisen und die von `Control` geerbte Eigenschaft `Dock` auf einen Wert der `DockStyle`-Aufzählung setzen. Wichtig ist dann nur noch, dass die `TopLevel`-Eigenschaft des Formulars auf `false` eingestellt wird:

```
ChildForm f = new ChildForm();
f.TopLevel = false;
this.Controls.Add(f);
f.Dock = DockStyle.Left;
f.Show();
```

Listing 429: Anzeigen eines andockenden Formulars

Das Beispiel erzeugt eine neue Instanz der Klasse `ChildForm`, die ein ganz normales Formular (mit einer `TextBox`) ist und im `Click`-Ereignis des Datei/Neu-Menüs programmiert.

Wenn Sie nun noch über das Rezept 225 dafür sorgen, dass der Anwender das Formular nicht bewegen kann, bleiben die Formulare immer schön aneinander und am Parent-Formular angedockt.

227 Ändern des Systemmenüs – Entfernen des Schließen-Eintrags und -Schalters

Den Schließen-Eintrag und den entsprechenden Schalter im Systemmenü bzw. in der Titelleiste eines Formulars können Sie nicht separat abschalten. Die Eigenschaft `ControlBox` erlaubt nur das komplette Abschalten des Systemmenüs. Wenn Sie aber ein Formular mit Systemmenü und Minimieren- und Maximieren-Schaltfläche, aber ohne den Schließen-Befehl benötigen, müssen Sie das Systemmenü über API-Funktionen verändern. Die Vorgehensweise entspricht der des Rezepts 225, nur dass Sie hier eben den Schließen-Befehl über die Kostante `SC_CLOSE` löschen.

Listing 430 implementiert dieses Löschen im `Load`-Ereignis eines Formulars. Neben dem Schließen-Befehl wird auch der Trennbalken entfernt, der vor dem Schließen-Eintrag platziert ist. Da dieser nicht über eine Id erreichbar ist, löscht `StartForm_Load` den Balken über dessen Position.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` einbinden.

```
[DllImport("user32.dll")]
private static extern IntPtr GetSystemMenu(IntPtr hWnd, int bRevert);

[DllImport("user32.dll")]
private static extern int DeleteMenu(IntPtr hMenu, int uPosition, int uFlags);
```

Listing 430: Entfernen des Schließen-Befehls aus dem Systemmenü

```
private const int SC_CLOSE = 0xF060;
private const int MF_BYCOMMAND = 0;
private const int MF_BYPOSITION = 0x00000400;

private void StartForm_Load(object sender, System.EventArgs e)
{
   // Handle des Systemmenüs ermitteln und den Schließen-Befehl und
   // den nun überflüssigen Separator löschen
   IntPtr sysMenuHandle = GetSystemMenu(this.Handle, 0);
   DeleteMenu(sysMenuHandle, SC_CLOSE, MF_BYCOMMAND);
   DeleteMenu(sysMenuHandle, 5, MF_BYPOSITION);
}
```

Listing 430: Entfernen des Schließen-Befehls aus dem Systemmenü (Forts.)

Das Löschen des Trennbalkens funktioniert auf diese Weise natürlich nur korrekt, sofern das Systemmenü nicht bereits vorher geändert wurde. Gegebenenfalls müssen Sie den Index (per Voreinstellung 5) anpassen.

228 Formulare verlaufend füllen

In einigen Anwendungen werden Formulare in einer Farbe verlaufend gefüllt dargestellt. Dieses »Feature« können Sie auch recht einfach programmieren ohne dafür ein Hintergrundbild verwenden zu müssen. GDI+ stellt dazu im Namensraum `System.Drawing.Drawing2D` die Klasse `LinearGradientBrush` zur Verfügung. Bei der Erzeugung eines solchen Pinsels geben Sie in einer Variante des Konstruktors das Ausgaberechteck an (damit der Verlauf korrekt berechnet werden kann), eine Startfarbe, eine Endfarbe und den Winkel der Füllrichtung. Im `Paint`-Ereignis können Sie einen `LinearGradientBrush`-Pinsel verwenden, um über `FillRectangle` ein gefülltes Rechteck zu zeichnen.

Damit bei einer Größenänderung des Formulars zum einen korrekt und zum anderen flackerfrei gezeichnet wird, sollten Sie im `Load`-Ereignis des Formulars zusätzlich über die `SetStyle`-Methode den Stil so einstellen, dass das Zeichnen in einem (optimierten) doppelten Puffer erfolgt und dass bei einer Größenänderung automatisch neu gezeichnet wird. Dieser »Trick« kommt eher aus der Steuerelement-Programmierung, kann aber (natürlich) auch für Formulare angewendet werden:

```
private void StartForm_Load(object sender, EventArgs e)
{
   // Flackern verhindern und Neuzeichnen bei einer Größenänderung erzwingen
   this.SetStyle(ControlStyles.OptimizedDoubleBuffer |
      ControlStyles.AllPaintingInWmPaint | ControlStyles.ResizeRedraw, true);
}

private void StartForm_Paint(object sender,
```

Listing 431: Flackerfreies verlaufendes Füllen eines Formulars

```
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // LinearGradientBrush für die verlaufende Füllung erzeugen
    LinearGradientBrush brush = new LinearGradientBrush(this.ClientRectangle,
        Color.Blue, Color.Black, 30, false);

    // Formularfläche mit dem Pinsel füllen
    g.FillRectangle(brush, this.ClientRectangle);
}
```

Listing 431: Flackerfreies verlaufendes Füllen eines Formulars (Forts.)

Das Beispiel benötigt neben den für eine Windows.Forms-Anwendung verwendeten Namensräumen den Import der Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D`.

Abbildung 126 zeigt das Ergebnis.

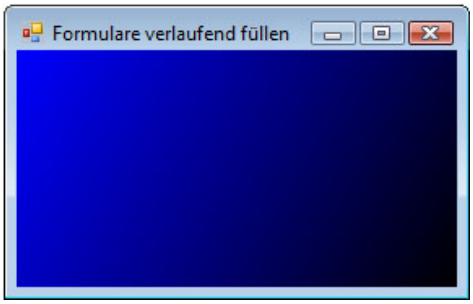


Abbildung 126: Ein mit einem Winkel von 30 Grad verlaufend gefülltes Formular

229 Formulare mit speziellen Formen

Formulare mit speziellen Formen (»Shaped Forms«) wie z. B. die Fenster, die mit vielen Multimedia-Anwendungen wie dem Windows Media Player und dem Winamp über deren Skins möglich sind, sind in einer Windows.Forms-Anwendung ebenfalls möglich. Grundlage dazu ist, dem Formular ein Bild zuzuweisen, das der späteren Form entspricht, oder auf dem Formular eine Grafik zu zeichnen. Zur Erstellung der besonderen Form können Sie zwei Techniken verwenden: einen transparenten Hintergrund und eine Region.

Zum Kompilieren der in diesem Rezept beschriebenen Methoden benötigen Sie neben dem Formular und der Referenz auf die Assembly *System.Forms.dll* zwei Bitmaps, die Sie der Ressource der Anwendung zuweisen. Unter Visual Studio fügen Sie die Bilder über das Ressourcen-Register in den Eigenschaften der Anwendung hinzu. Daneben müssen Sie die Namensräume `System`, `System.Drawing`, `System.Drawing.Drawing2D` und `System.Reflection` importieren.

Transparenter Hintergrund

Die einfachste Lösung zur Erstellung eines Formulars mit einer speziellen Form ist, einen einfarbigen Hintergrund zu verwenden und der Eigenschaft `TransparencyKey` des Formulars diese Hintergrundfarbe zuzuweisen. Stellen Sie dann noch die Eigenschaft `FormBorderStyle` auf `FormBorderStyle.None`, kann das Ergebnis sich so einigermaßen sehen lassen.

Abbildung 127 zeigt ein solches Formular, dessen `BackgroundColor`- und `TransparencyKey`-Eigenschaft auf `Color.Silver` eingestellt ist.

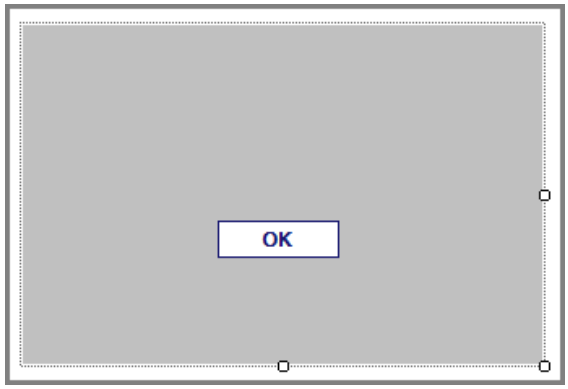


Abbildung 127: Das Basisformular in der Entwicklungsumgebung

Der Schalter ist übrigens ein ganz normaler `Button` im flachen Stil, dem als Hintergrundbild dieselbe Textur zugewiesen wurde, die auch für das später gezeichnete Rechteck verwendet wird.

Im `Paint`-Ereignis des Formulars werden ein gefülltes Rechteck und ein gefüllter Kreis gezeichnet:

```
private void ShapedForm1_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
   Graphics g = e.Graphics;

   // Anti-Aliasing einstellen
   g.SmoothingMode = SmoothingMode.HighQuality;

   // Die beiden Texturen aus der Ressource lesen, damit zwei
   // TextureBrush-Objekte für das Zeichnen erzeugen und
   // ein gefülltes Rechteck und einen gefüllten Kreis zeichnen
   using (TextureBrush brush1 =
      new TextureBrush(Properties.Resources.Texture1))
   {
      g.FillRectangle(brush1, 0, 40, 300, 100);
   }
```

Listing 432: Zeichnen eines mit einer Textur gefüllten Rechtecks und Kreises

```
    using (TextureBrush brush2 =
       new TextureBrush(Properties.Resources.Texture2))
    {
       g.FillEllipse(brush2, 65, 5, 180, 180);
    }
}
```

Listing 432: Zeichnen eines mit einer Textur gefüllten Rechtecks und Kreises (Forts.)

Hinweis

Die Bitmap-Ressourcen habe ich dem Projekt über den Ressourcen-Designer von Visual Studio hinzugefügt, weswegen das Beispiel diese über die vom Designer erstellte Klasse `Resources` typsicher einlesen kann.

Abbildung 128 zeigt das Ergebnis.



Abbildung 128: Ein Formular mit transparentem Hintergrund

Diese Technik führt zwar zu einem schnellen Ergebnis, ist allerdings nicht optimal: Alle Formularbereiche in der als transparent definierten Farbe erscheinen ebenfalls durchsichtig. Wäre der Schalter auf dem Formular in Abbildung 128 ein normaler Schalter, wären die silbernen Teile des Schalters ebenfalls durchsichtig. Besonders schön sieht das dann nicht mehr aus. Das Problem können Sie etwas mildern, indem Sie eine sehr spezielle Farbe als transparente Farbe einsetzen. Die Gefahr, dass diese Farbe in den Steuerelementen oder Zeichnungen des Formulars vorkommt, ist aber dennoch vorhanden.

Regionen

Die Lösung dieses Problems ist eine *Region*. Eine solche definiert den sichtbaren Fensterbereich eines Formulars (oder eines Steuerelements) und kann in beliebigen Formen erstellt werden. Die Region eines Formulars erreichen Sie über dessen Eigenschaft `Region`. Per Default ist einem Formular (logischerweise) keine Region zugewiesen, weswegen die Eigenschaft zunächst den Wert `null` speichert.

Sie können aber eigene Regionen erzeugen und dem Formular zuweisen. Beim Erzeugen eines `Region`-Objekts (aus dem Namensraum `System.Drawing`) können Sie u. a. ein `Rectangle`-Objekt

übergeben (das eine einfache, rechteckige Region definiert). Für komplexere Regionen übergeben Sie dem Konstruktor jedoch ein `GraphicsPath`-Objekt. Die `GraphicsPath`-Klasse (aus dem Namensraum `System.Drawing.Drawing2D`) erlaubt über eine Vielzahl an Methoden die Definition der Region. Die Methode `AddLine` fügt dem Pfad z. B. eine Linie hinzu, `AddArc` fügt einen Ellipsen-Ausschnitt hinzu, `AddRectangle` ein Rechteck, `AddEllipse` eine Ellipse oder einen Kreis etc. Diese Methoden sind funktional identisch zu den entsprechenden *Draw*-Methoden der `Graphics`-Klasse, was deren Anwendung erheblich vereinfacht. Das Geniale an diesen Methoden ist, dass Linien oder Ellipsenbögen, die sich an den Endpunkten berühren, automatisch verbunden werden. So können Sie mit ein wenig Arbeit auch die komplexesten Formen realisieren.

Listing 433 zeigt die Definition einer Region für das Formular in Abbildung 128. Die Region wird im `Load`-Ereignis des Formulars erzeugt. Das Beispiel ist bewusst etwas umständlich (wie Sie in Listing 9 sehen, geht es in unserem Fall auch etwas einfacher). Daran können Sie erkennen, wie komplexe Regionen erzeugt werden können.

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
   // Neue Region über ein GraphicsPath-Objekt definieren
   GraphicsPath path = new GraphicsPath();

   // Obere Linie mit Kreis-Halbbogen
   path.AddLine(0, 40, 85, 40);
   path.AddArc(65, 5, 180, 180, 218, 104);
   path.AddLine(225, 40, 300, 40);

   // Untere Linie mit Kreis-Halbbogen
   path.AddLine(0, 140, 78, 140);
   path.AddArc(65, 5, 180, 180, 30, 120);
   path.AddLine(232, 140, 300, 140);

   // Linke und rechte Linie
   path.AddLine(0, 40, 0, 140);
   path.AddLine(300, 40, 300, 140);

   // Region dem Formular zuweisen
   this.Region = new Region(path);
}
```

Listing 433: Definieren der Region des Formulars über ein GraphicsPath-Objekt

Als kleinen Trick habe ich die Region übrigens zuvor zunächst im `Paint`-Ereignis testweise simuliert, indem ich statt `path.`*Add* einfach `e.Graphics.`*Draw* eingesetzt und am ersten Argument der `Draw`-Methoden ein `Pen`-Objekt eingefügt habe:

```
private void ShapedForm2_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
   Graphics g = e.Graphics;

   // Obere Linie mit Kreis-Halbbogen
   Pen p = new Pen(Color.Black, 2);
   g.DrawLine(p, 0, 40, 85, 40);
   g.DrawArc(p, 65, 5, 180, 180, 218, 104);
   g.DrawLine(p, 225, 40, 300, 40);

   // Untere Linie mit Kreis-Halbbogen
   g.DrawLine(p, 0, 140, 78, 140);
   g.DrawArc(p, 65, 5, 180, 180, 30, 120);
   g.DrawLine(p, 232, 140, 300, 140);

   // Linke und rechte Linie
   g.DrawLine(p, 0, 40, 0, 140);
   g.DrawLine(p, 300, 40, 300, 140);
}
```

Listing 434: Testweise Simulation einer Region im Paint-Ereignis

Das Ergebnis dieses Tests sehen Sie in Abbildung 129.

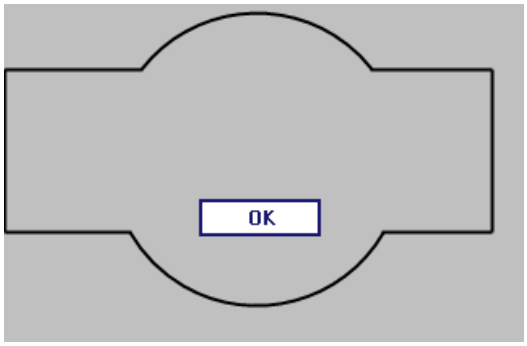


Abbildung 129: Testweises Zeichnen der Region im Paint-Ereignis

Die eigentliche `Paint`-Methode sieht dann so aus:

```
private void ShapedForm2_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
   Graphics g = e.Graphics;
```

Listing 435: Zeichnen eines Rechtecks und eines Kreises über einer Region

```
   // Anti-Aliasing einstellen
   g.SmoothingMode = SmoothingMode.HighQuality;

   // Die beiden Texturen aus der Ressource lesen, damit zwei
   // TextureBrush-Objekte für das Zeichnen erzeugen und
   // ein gefülltes Rechteck und einen gefüllten Kreis zeichnen
   using (TextureBrush brush1 =
      new TextureBrush(Properties.Resources.Texture1))
   {
      g.FillRectangle(brush1, 0, 40, 300, 100);
   }
   using (TextureBrush brush2 =
      new TextureBrush(Properties.Resources.Texture2))
   {
      g.FillEllipse(brush2, 65, 5, 180, 180);
   }
}
```

Listing 435: Zeichnen eines Rechtecks und eines Kreises über einer Region (Forts.)

Das Ergebnis des Programms entspricht zusammen mit dem Zeichnen im `Paint`-Ereignis der Abbildung 128, nur dass eben hier die Probleme mit der transparenten Farbe und dem Klicken auf dem transparenten Formularhintergrund vermieden werden.

Wie bereits gesagt können Sie die Region in unserem Fall auch einfacher definieren. Dazu fügen Sie dem Pfad einfach ein Rechteck und einen Kreis hinzu, der den im `Paint`-Ereignis gezeichneten entspricht. Dabei kommt ein weiteres interessantes Feature von Regionen ins Spiel: Per Voreinstellung werden geschlossene Pfade, die sich überlagern, alternierend unsichtbar/sichtbar geschaltet. Auf diese Weise können Sie auch Formen erzeugen, die im Innenbereich transparent sind. Die Definition eines Rechtecks mit der gefolgten Definition eines Kreises würde den Bereich, an dem der Kreis das Rechteck überschneidet, durchsichtig erscheinen lassen. Wenn Sie dieses Verhalten ändern wollen, geben Sie am ersten Argument des `GraphicsPath`-Konstruktors den Wert `FillMode.Winding` an.

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
   GraphicsPath path = new GraphicsPath(FillMode.Winding);

   path.AddRectangle(new Rectangle(0, 40, 300, 100));
   path.AddEllipse(65, 5, 180, 180);

   this.Region = new Region(path);
}
```

Listing 436: Einfachere Definition einer Region mit geometrischen Formen

230 Formulare mit dem Vista-Glas-Effekt ausstatten

Windows Vista zeigt im Aero-Thema den Rahmen von Fenstern im Aero-Glas-Effekt an. Für manche Anwendungen wäre es wünschenswert, nicht nur den Default-Rahmen, sondern einen verbreiterten Rahmen oder das gesamte Fenster in diesem Effekt anzuzeigen. Abbildung 130 zeigt ein Beispiel.

Die Lösung für dieses Problem liegt in zwei API-Funktionen: `DwmIsCompositionEnabled` und `DwmExtendFrameIntoClientArea`. `DwmIsCompositionEnabled` überprüft, ob die für dieses Feature notwendige »Desktopgestaltung« (Desktop composition) von Vista zurzeit eingeschaltet ist. Diese ist u. a. für den Glas-Effekt notwendig und kann auch in den erweiterten Systemeinstellungen abgeschaltet werden. `DwmExtendFrameIntoClientArea` erweitert den Rahmen des Fensters in den Clientbereich hinein. Dabei können Sie die Breite der Erweiterung für jede Seite einzeln über die `MARGINS`-Struktur angeben, die Sie am zweiten Argument übergeben. So können Sie auch Fenster erzeugen, die einen breiteren (Glas-)Rahmen besitzen. Wenn Sie -1 in allen Feldern der `MARGINS`-Struktur angeben, wird der Rahmen in den kompletten Innenbereich des Fensters verbreitert.

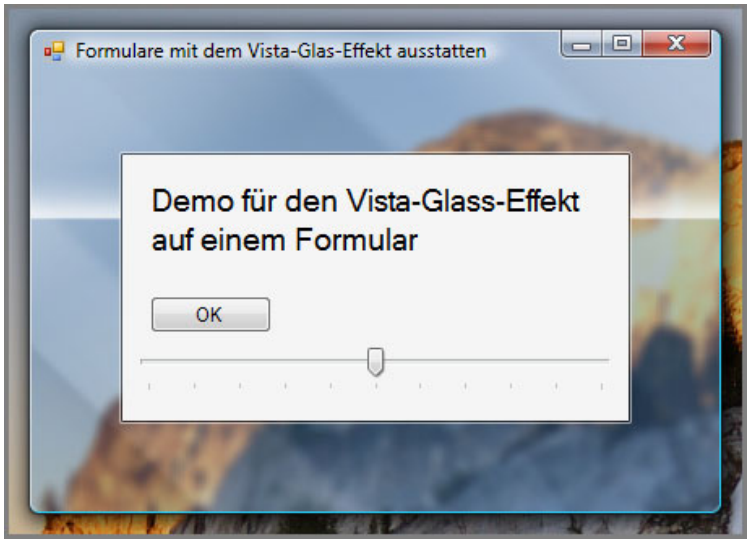


Abbildung 130: Beispiel für ein Windows.Forms-Formular mit erweitertem Vista-Glass-Rahmen

Leider funktioniert das Ganze unter `Windows.Forms` nicht besonders gut (unter WPF schon, wie das entsprechende Rezept im WPF-Kapitel zeigt). Das Problem ist, dass Vista per Voreinstellung die Farbe Schwarz als Glas-Farbe einsetzt. Um den Rahmen im Aero-Thema korrekt verglast vergrößern zu können, muss der Hintergrund des Fensters schwarz sein. Das ist schon einmal deswegen dumm, weil Sie den Rahmen nicht ohne weiteres einfach nur verbreitern können, wenn der Clientbereich nicht schwarz sein soll.

Wenn Sie den Rahmen so vergrößern, dass dieser den gesamten Innenbereich des Fensters belegt, sieht das Fenster selbst zwar gut aus. Das Problem ist nur, dass alle schwarzen Bereiche von Steuerelementen, die auf dem Fenster angelegt sind, auch im Glas-Effekt angezeigt werden. Und das sieht sehr unschön aus.

Im Internet kursieren Lösungsansätze, die die `TransparentKey`-Eigenschaft des Formulars einsetzen und das Fenster damit transparent schalten. Diese haben aber den Nachteil, dass der Anwender durch das transparente Fenster hindurchklicken kann (z. B. auf den Desktop dahinter).

Meine Lösung des Problems habe ich in Form der Klasse `GlassForm` implementiert, von der Sie Ihre Formulare ableiten können. Zum Kompilieren dieser Klasse müssen Sie die Namensräume `System`, `System.Drawing`, `System.Runtime.InteropServices` und `System.Windows.Forms` einbinden.

```
public class GlassForm : Form
{
   /* Struktur, die bestimmt, um wie viele Pixel der Rahmen
      zu jeder Seite verbreitert wird */
   [StructLayout(LayoutKind.Sequential)]
   public struct MARGINS
   {
      public int Left;
      public int Right;
      public int Top;
      public int Bottom;
   }

   /* Verbreitert den Vista-Rahmen nach innen */
   [DllImport("dwmapi.dll", PreserveSig = false)]
   static extern void DwmExtendFrameIntoClientArea(IntPtr hwnd,
      ref MARGINS margins);

   /* Überprüft, ob die Desktopgestaltung aktiviert ist */
   [DllImport("dwmapi.dll", PreserveSig = false)]
   static extern bool DwmIsCompositionEnabled();

   /* Panel für den Inhalt */
   protected Panel contentPanel;

   /* Die Hintergrundfarbe */
   public override Color BackColor
   {
      get
      {
         return this.contentPanel.BackColor;
      }
      set
      {
         this.contentPanel.BackColor = value;
      }
   }
```

Listing 437: Basisklasse für ein Formular, dessen Rahmen unter Vista vergrößert werden kann

```
/* Überprüft, ob ein Glass-Rahmen möglich ist */
public bool IsGlassFrameEnabled
{
   get
   {
      try
      {
         return DwmIsCompositionEnabled();
      }
      catch (DllNotFoundException)
      {
         // Die DLL dwmapi.dll ist nicht verfügbar.
         // Wahrscheinlich läuft die Anwendung unter
         // einer älteren Windows-Version
         return false;
      }
   }
}

/* Konstruktor */
public GlassForm()
{
   // Panel erzeugen und den Steuerelementen hinzufügen
   this.contentPanel = new Panel();
   this.contentPanel.Left = 0;
   this.contentPanel.Top = 0;
   this.contentPanel.Width = this.ClientRectangle.Width;
   this.contentPanel.Height = this.ClientRectangle.Height;
   this.contentPanel.Anchor = AnchorStyles.Left |
      AnchorStyles.Top | AnchorStyles.Right |
      AnchorStyles.Bottom;
   this.contentPanel.BackColor = base.BackColor;
   this.SuspendLayout();
   this.Controls.Add(this.contentPanel);
   this.ResumeLayout(false);
}

/* Platziert das Panel neu, wenn die Padding-Eigenschaft geändert wird */
protected override void OnPaddingChanged(EventArgs e)
{
   base.OnPaddingChanged(e);

   // Die Position des Panels neu berechnen
   this.contentPanel.Left = this.Padding.Left;
   this.contentPanel.Top = this.Padding.Top;
```

Listing 437: Basisklasse für ein Formular, dessen Rahmen unter Vista vergrößert werden kann (Forts.)

```
      this.contentPanel.Width = this.ClientRectangle.Width -
         this.Padding.Left - this.Padding.Right;
      this.contentPanel.Height = this.ClientRectangle.Height -
         this.Padding.Top - this.Padding.Bottom;

      // Den Rahmen neu definieren
      this.ExtendFrame();
   }

   /* Verbreitert den Rahmen des Fensters entsprechend den Einstellungen
      in Padding, wenn das Formular geladen wird */
   protected override void OnLoad(EventArgs e)
   {
      base.OnLoad(e);

         this.ExtendFrame();
   }

   /* Zeichnet den Hintergrund schwarz um den Glass-Effekt zu ermöglichen */
   protected override void OnPaintBackground(PaintEventArgs e)
   {
      base.OnPaintBackground(e);
      if (this.IsGlassFrameEnabled)
      {
         // Die Hintergrundfarbe auf Schwarz setzen um den
         // Glass-Effekt zu ermöglichen
         e.Graphics.Clear(Color.Black);
      }
   }

   /* Wird überschrieben um auf den Wechsel des Vista-Themas zu reagieren */
   protected override void WndProc(ref Message m)
   {
      const int DWMCOMPOSITIONCHANGED = 0x031E;
      if (m.Msg == DWMCOMPOSITIONCHANGED)
      {
         // Den Rahmen neu definieren
         this.ExtendFrame();
         this.Invalidate();
      }
      base.WndProc(ref m);
   }
```

Listing 437: Basisklasse für ein Formular, dessen Rahmen unter Vista vergrößert werden kann (Forts.)

```
   /* Verbreitert den Rahmen des Fensters entsprechend
      den Einstellungen in Padding */
   private void ExtendFrame()
   {
      if (this.IsGlassFrameEnabled)
      {
         if (this.DesignMode == false)
         {
            // Wenn nicht im Design-Modus: den Rahmen verbreitern
            MARGINS margins = new MARGINS()
            {
               Left = this.Padding.Left,
               Top = this.Padding.Top,
               Right = this.Padding.Right,
               Bottom = this.Padding.Bottom
            };
            DwmExtendFrameIntoClientArea(this.Handle, ref margins);
         }
      }
   }
}
```

Listing 437: Basisklasse für ein Formular, dessen Rahmen unter Vista vergrößert werden kann (Forts.)

`GlassForm` verbreitert den Rahmen des Fensters entsprechend der `Padding`-Eigenschaft des Formulars. Um die Probleme mit der Hintergrundfarbe zu lösen setzt `GlassForm` einen kleinen Trick ein: Das Formular verwendet ein `Panel`, das automatisch entsprechend der `Padding`-Eigenschaft positioniert wird. Damit haben Sie im Designer von Visual Studio die Möglichkeit, Ihre Steuerelemente auf dem `Panel` zu platzieren (und mit der `Anchor`-Eigenschaft anzuheften). Die `BackColor`-Eigenschaft des Formulars wird an dieses Panel delegiert, damit es die eigentlich eingestellte Hintergrundfarbe des Formulars anzeigt.

Die Eigenschaft `IsGlassFrameEnabled` überprüft, ob die notwendige Desktopgestaltung aktiviert ist. Dabei wird gleich die `DllNotFoundException` abgefangen, die geworfen wird, wenn die DLL *dwmapi.dll*, die die genannten API-Funktionen enthält, nicht gefunden wird (weil die Anwendung auf einer älteren Windows-Version ausgeführt wird). Diese Eigenschaft wird intern an verschiedenen Stellen eingesetzt um zu erreichen, dass das Formular auch ausgeführt wird, wenn die Desktopgestaltung nicht aktiviert ist oder das Programm unter einer älteren Windows-Version läuft.

In der überschriebenen Methode `OnPaintBackground` wird der Hintergrund des Formulars schwarz gezeichnet, um den Glass-Effekt zu ermöglichen. Die Methode `ExtendFrame`, die in `OnLoad` und in der überschriebenen `Padding`-Eigenschaft aufgerufen wird, verbreitert den Vista-Rahmen (wenn `IsGlassFrameEnabled true` zurückgibt). `ExtendFrame` stellt schließlich die `MARGINS`-Struktur zusammen und übergibt diese der `DwmExtendFrameIntoClientArea`-Methode.

Zusätzlich überschreibt GlassForm noch die WndProc-Methode, um den Wechsel des Vista-Themas (bzw. der Desktopgestaltung) abzufangen und das Fenster neu zu initialisieren (falls der Anwender das Vista-Thema wechselt, während die Anwendung ausgeführt wird).

Ich habe GlassForm unter Vista im Aero-Thema, im Vista-Basis-Thema und im Klassik-Thema und unter XP getestet. Dabei traten keine Problem auf. Ist Aero nicht aktiviert, ist es u. U. unschön, dass der Innenraum des Formulars trotzdem den in Padding angegebenen Rand darstellt. Das ließ sich aber nicht verhindern.

231 Splash-Formulare

Splash-Formulare sind Formulare, die beim Starten einer Anwendung erscheinen, um den Anwender während einer länger andauernden Initialisierungsphase Informationen anzubieten oder um einfach nur anzuzeigen, dass das Programm im Moment initialisiert wird.

Wenn Sie die Initialisierungen im Load-Ereignis des Startformulars vornehmen, ist die Anzeige eines Splash-Formulars sehr einfach. Listing 438 zeigt dies am Beispiel. Das Splash-Formular mit dem Namen *SplashForm* wird in diesem Beispiel nach dem Erzeugen und Anzeigen in einer Schleife aktualisiert. Dazu schreibt StartForm_Load den aktuellen Indexwert in ein Label mit dem Namen *infoLabel*, das auf dem Splash-Formular angelegt ist, und ruft die Refresh-Methode dieses Labels auf um die Anzeige zu aktualisieren.

Der Aufruf von Application.DoEvents nach dem Anzeigen des Splash-Formulars ist übrigens sehr wichtig. Ohne diesen Aufruf würde das Formular – besonders wenn es Bilder enthält – nicht unbedingt vor der Schleife vollständig gezeichnet werden. Der Grund dafür liegt in der Tatsache, dass die Priorität von Ausgaben auf einem Fenster in der Windows-Nachrichtenwarteschleife nur sehr gering ist. Ohne DoEvents würde das Programm die Schleife schon abarbeiten, während Windows das Fenster noch gar nicht zu Ende gezeichnet hat. DoEvents zwingt Windows hingegen, seine Nachrichtenwarteschleife komplett abzuarbeiten.

Die Ausnahmebehandlung in Listing 438 ist für die Praxis ebenfalls sehr wichtig. Falls beim Initialisieren Fehler auftreten, sollte die Anwendung normalerweise nicht starten. Das Beispiel ruft deswegen im Fehlerfall Application.Exit auf.

```
private void StartForm_Load(object sender, System.EventArgs e)
{
   // Splash-Formular erzeugen und anzeigen
   SplashForm f = new SplashForm();
   f.Show();
   Application.DoEvents();

   // Simulation einer Initialisierung
   try
   {
      for (int i = 0; i < 100; i++)
      {
```

Listing 438: Anzeigen eines Splash-Formulars im Load-Ereignis des Startformulars

```
            f.infoLabel.Text = "Lese Datensätze. Datensatz " + i;
            f.infoLabel.Refresh();

            Thread.Sleep(30);
         }
      }
      catch (Exception ex)
      {
         MessageBox.Show("Fehler beim Initialisieren: " + ex.Message,
            Application.ProductName, MessageBoxButtons.OK,
            MessageBoxIcon.Error);
         Application.Exit();
      }

      // Splash-Formular wieder schließen
      f.Close();
   }
```

Listing 438: Anzeigen eines Splash-Formulars im Load-Ereignis des Startformulars (Forts.)

Zum Kompilieren dieser Methode benötigen Sie neben der Referenz auf die Assembly *System.Windows.Forms.dll* den Import der Namensräume `System`, `System.Windows.Forms` und `System.Threading`.

Abbildung 131 zeigt das Beispiel-Splash-Formular beim Start der Anwendung.



Abbildung 131: Splash-Formular beim Start einer Anwendung

Auf eine ähnliche Weise können Sie natürlich auch in der `Main`-Methode der Anwendung initialisieren und das Splash-Formular anzeigen:

```
[STAThread]
static void Main()
{
```

Listing 439: Anzeigen eines Splash-Formulars in der Main-Methode

```
   Application.EnableVisualStyles();

   // Splash-Formular erzeugen und anzeigen
   SplashForm f = new SplashForm();
   f.Show();
   Application.DoEvents();

   // Simulation einer Initialisierung
   try
   {
      for (int i = 0; i < 100; i++)
      {
         f.infoLabel.Text = "Lese Datensatz " + i + " ...";
         f.infoLabel.Refresh();

         System.Threading.Thread.Sleep(30);
      }
   }
   catch (Exception ex)
   {
      MessageBox.Show("Fehler beim Initialisieren: " + ex.Message,
         Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Error);
      Application.Exit();
   }

   // Splash-Formular wieder schließen
   f.Close();

   // Die Anwendung starten
   Application.Run(new StartForm());
}
```

Listing 439: Anzeigen eines Splash-Formulars in der Main-Methode (Forts.)

232 Ermitteln, auf welchen Monitoren ein Formular angezeigt wird

In einigen Fällen ist es notwendig, Informationen über den Monitor einzulesen, auf dem ein Formular angezeigt wird. Das ist zum Beispiel dann der Fall, wenn das Formular in einer bestimmten, auf die aktuelle Auflösung bezogenen Größe angezeigt werden soll. Ist an das System nur ein Monitor angeschlossen, können Sie diese Informationen über die Eigenschaft `PrimaryScreen` der `Screen`-Klasse aus dem Namensraum `System.Windows.Forms` ermitteln.

Viele (meist professionelle) Anwender haben jedoch mittlerweile mehrere Bildschirme an ihren Rechner angeschlossen. In diesem Fall kann ein Formular auch auf verschiedenen Monitoren angezeigt werden. Und es kann sogar über mehrere Monitore verteilt sein.

Um das Problem der Ermittlung der Monitore, auf denen ein Formular angezeigt wird, zu lösen habe ich die Methode `GetScreensOfForm` entwickelt. Diese Methode erwartet am einzigen Argument eine Referenz auf ein Formular. `GetScreensOfForm` gibt eine `ReadOnlyCollection` (aus dem Namensraum `System.Collections.ObjectModel`) zurück, die für jeden Monitor, auf dem das Formular angezeigt wird, eine `Screen`-Instanz referenziert.

Bei der Entwicklung von `GetScreensOfForm` habe ich die Tatsache genutzt, dass Windows mehrere angeschlossene Monitore zu einem virtuellen Monitor zusammensetzt. Der linke Monitor beginnt meist an der Position (0, 0), kann aber auch an einer negativen X-Position beginnen (nämlich dann, wenn der rechte Monitor der primäre ist). Die Position des jeweils rechten Monitors beginnt an der X-Position des linken addiert mit seiner Auflösungs-Breite (in Pixel). Besitzt das System z. B. einen primären Monitor mit der Auflösung 1280 * 1024 und rechts davon einen sekundären Monitor mit der Auflösung 1024 * 768, beginnt der linke Monitor an der X-Position 0 und der rechte an der X-Position 1281. Die Y-Position der sekundären Monitore ist übrigens abhängig davon, wie diese in den Anzeige-Einstellungen von Windows platziert wurden.

Informationen über die an das System angeschlossenen Monitore erhalten Sie über die `AllScreens`-Eigenschaft der `Screen`-Klasse. Theoretisch können Sie diese einfach durchgehen und die Bildschirme, auf denen ein Formular liegt, über die Position und Breite des Formulars und die Position und Breite des jeweiligen Bildschirms berechnen. Dummerweise ist das `Screen`-Array, das `AllScreens` verwaltet, aber nicht nach der X-Position der Bildschirme sortiert. Das liegt wahrscheinlich daran, dass `AllScreens[0]` scheinbar (und undokumentiert) immer den primären Bildschirm referenziert. Liegt links von diesem ein sekundärer, so beginnt dieser an einer negativen X-Position, wird aber in `AllScreens[1]` referenziert.

Um die Positionen der einzelnen Bildschirme korrekt berücksichtigen zu können müssen diese also nach der X-Position sortiert werden. Dazu habe ich den von `AllScreens` zurückgegebenen Array geklont und die `Sort`-Methode der `Array`-Klasse mit einem Lambda-Ausdruck aufgerufen, der den Vergleich der `Screen`-Instanzen übernimmt. Der Lambda-Ausdruck entspricht dem Delegaten `Comparison<Screen>`, der der `Sort`-Methode am zweiten Argument übergeben werden kann.

Der Rest der Methode ist lediglich ein wenig Rechenarbeit, um herauszufinden, auf welchen Monitoren das Formular angezeigt wird. Ich denke, die Kommentare sprechen für sich.

Zum Kompilieren der `GetScreensOfForm`-Methode müssen Sie die Namensräume `System`, `System.Collections.Generic`, `System.Collections.ObjectModel` und `System.Windows.Forms` importieren.

```
/* Ermittelt den bzw. die Screens, auf dem ein Formular liegt */
public static ReadOnlyCollection<Screen> GetScreensOfForm(Form form)
{
   // Auflistung für das Ergebnis erzeugen
   List<Screen> screensOfForm = new List<Screen>();

   if (Screen.AllScreens.Length == 1)
   {
```

Listing 440: Klasse und Methode zur Ermittlung der Bildschirme, auf denen ein Formular angezeigt wird

```
      // Wenn nur ein Monitor existiert, kann das Formular
      // nur auf diesem angezeigt werden
      screensOfForm.Add(Screen.PrimaryScreen);
   }
   else
   {
      // Screens nach der X-Position sortieren
      Screen[] screens = (Screen[])Screen.AllScreens.Clone();
      Array.Sort(screens, (x, y) => x.Bounds.Left.CompareTo(y.Bounds.Left));

      // Ermitteln des linken Bildschirms
      int leftScreenIndex = 0;
      for (int i = 0; i < screens.Length; i++)
      {
         if (form.Left <= screens[i].Bounds.Left + screens[i].Bounds.Width)
         {
            leftScreenIndex = i;
            break;
         }
      }

      // Ermitteln des rechten Bildschirms
      int rightScreenIndex = -1;
      for (int i = 0; i < screens.Length; i++)
      {
         int x = form.Left + form.Width;
         if (x >= screens[i].Bounds.Left && x <=
            screens[i].Bounds.Left + screens[i].Bounds.Width)
         {
            rightScreenIndex = i;
            break;
         }
      }
      if (rightScreenIndex == -1)
      {
         // Wenn der rechte Bildschirm nicht ermittelt werden konnte,
         // geht das Formular über diesen hinaus
         rightScreenIndex = screens.Length - 1;
      }

      // Überprüfen, ob das Formular über mehrere Bildschirme geht
      if (leftScreenIndex != rightScreenIndex)
      {
         // Den linken Screen ablegen
         screensOfForm.Add(screens[leftScreenIndex]);
```

Listing 440: Klasse und Methode zur Ermittlung der Bildschirme, auf denen ein Formular angezeigt wird (Forts.)

```
            // Die weiteren Bildschirme, die gegebenenfalls zwischen dem
            // linken und dem rechten liegen, hinzufügen
            for (int i = leftScreenIndex + 1; i < rightScreenIndex; i++)
            {
               screensOfForm.Add(screens[i]);
            }

            // Den rechten Bildschirm hinzufügen
            screensOfForm.Add(screens[rightScreenIndex]);
         }
         else
         {
            // Das Formular liegt lediglich auf einem Bildschirm
            screensOfForm.Add(screens[leftScreenIndex]);
         }
      }

      // Ergebnis zurückgeben
      return new ReadOnlyCollection<Screen>(screensOfForm);
   }
```

Listing 440: Klasse und Methode zur Ermittlung der Bildschirme, auf denen ein Formular angezeigt wird (Forts.)

Das Ergebnis auszuwerten ist natürlich unter Umständen etwas schwierig. Liegt ein Formular auf mehreren Bildschirmen, muss in vielen Fällen einer davon als der Haupt-Bildschirm angesehen werden. Ich habe in meinen Programmen in diesem Fall einfach den ersten (also den linken) Bildschirm verwendet. Alternativ können Sie auch ermitteln, auf welchem Bildschirm der Hauptteil des Formulars liegt. Oder Sie gehen von einem Steuerelement aus und berechnen über dessen X-Position den Bildschirm, auf dem dieses liegt:

```
ReadOnlyCollection<Screen> screensOfForm = GetScreensOfForm(this);
Control controlToCheck = ...
int controlX = this.PointToScreen(controlToCheck.Location).X;
Screen screenOfControl = null;
foreach (Screen screen in screensOfForm)
{
   if (controlX > screen.Bounds.X &&
      controlX < screen.Bounds.X + screen.Bounds.Width)
   {
      screenOfControl = screen;
      break;
   }
}
```

Listing 441: Ermitteln des Bildschirms, an dem ein Steuerelement beginnt

233 Ein Formular auf einem sekundären Bildschirm öffnen

In einer Windows-Anwendung werden Formulare standardmäßig auf dem primären Bildschirm geöffnet. Lediglich wenn ein Formular mit einem Besitzer erzeugt wird (der dem Formular am Konstruktor übergeben wird), wird dieses auf dem Bildschirm geöffnet, auf dem der größte Teil des Besitzer-Formulars liegt. In einigen Fällen wollen Sie aber vielleicht ein Formular explizit auf einem sekundären Bildschirm öffnen (natürlich nur, sofern das System über mehrere Bildschirme verfügt).

Die Methode `SetPositionToFirstSecondaryScreen` in Listing 442 stellt die Position des übergebenen Formulars auf die Mitte des ersten sekundären Bildschirms des Systems ein, oder – falls keine sekundären Bildschirme vorhanden sind – auf die Mitte des primären Bildschirms. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Windows.Forms` und `System.Drawing` importieren.

```
public static void SetPositionToFirstSecondaryScreen(Form form)
{
   // Das Argument überprüfen
   if (form == null)
   {
      throw new ArgumentException("form darf nicht null sein");
   }

   // Start-Position auf manuell stellen
   form.StartPosition = FormStartPosition.Manual;

   // Alle Bildschirme des Systems durchgehen um den ersten
   // sekundären zu finden
   Screen screen = Screen.PrimaryScreen;
   if (Screen.AllScreens.Length > 1)
   {
      // Sekundär-Bildschirme sind vorhanden: Suchen des ersten
      // Sekundär-Bildschirms
      for (int i = 0; i < Screen.AllScreens.Length; i++)
      {
         // Überprüfen, ob der aktuelle Bildschirm nicht der primäre ist
         if (Screen.AllScreens[i] != Screen.PrimaryScreen)
         {
            screen = Screen.AllScreens[i];
            break;
         }
      }
   }
```

Listing 442: Methode zum Platzieren eines Formulars auf dem ersten sekundären Bildschirm des Systems

```
   // Formular auf den gefundenen Bildschirm platzieren
   int x = screen.WorkingArea.Left +
      ((screen.WorkingArea.Width - form.Width) / 2);
   int y = screen.WorkingArea.Top +
      ((screen.WorkingArea.Height - form.Height) / 2);
   form.Location = new Point(x, y);
}
```

Listing 442: Methode zum Platzieren eines Formulars auf dem ersten sekundären Bildschirm des Systems (Forts.)

234 Die Bildschirm-Position eines Steuerelements ermitteln

In einigen Fällen (zum Beispiel in einer abgewandelten Form in Rezept 235 ☺) ist es notwendig, die Position eines Steuerelements bezogen auf den Bildschirm zu ermitteln. Prinzipiell können Sie dazu einfach die `PointToScreen`-Methode des Formulars unter der Übergabe der `Location`-Eigenschaft des Steuerelements aufrufen. Das funktioniert aber nicht, wenn das Steuerelement auf einem weiteren Container, wie z. B. einem `Panel` angelegt ist, denn die Position eines Steuerelements bezieht sich immer auf den Container. Sie müssen also alle Container, auf denen das Steuerelement direkt oder indirekt angelegt ist, berücksichtigen. Und genau das macht die Methode `GetScreenPositionOfControl` in Listing 443.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Windows.Forms` und `System.Drawing` importieren.


```
public static Point GetScreenPositionOfControl(Control control)
{
   //Die Position des Steuerelements auslesen
   int x = control.Location.X;
   int y = control.Location.Y;

   // Das Parent-Formular ermitteln
   Form parentForm = control.FindForm();
   if (parentForm != null)
   {
      // Alle Container durchgehen, auf denen das Steuerelement
      // direkt oder indirekt liegt
      Control parent = control.Parent;
      while (parent != parentForm)
      {
         // Wenn der aktuelle Container nicht das Formular ist
         // wird die Position des Containers zu der Position des
         // Steuerelements addiert
         x += parent.Location.X;
         y += parent.Location.Y;
```

Listing 443: Methode zur Ermittlung der Bildschirm-Position eines Steuerelements

```
            parent = parent.Parent;
         }

         // Die ermittelte Position in Bildschirm-Koordinaten zurückgeben
         return parentForm.PointToScreen(new Point(x, y));
      }
      else
      {
         throw new ArgumentException("Das übergebene Steuerelement " +
            "besitzt kein Parent-Formular");
      }
   }
```

Listing 443: Methode zur Ermittlung der Bildschirm-Position eines Steuerelements (Forts.)

235 Die optimale Position eines Formulars oder eines Steuerelements bezogen auf ein Steuerelement ermitteln

In einem meiner Projekte habe ich mehrere Steuerelemente entwickelt, die bei einem Klick auf einen rechts angebrachten Schalter ein Formular oder ein weiteres Steuerelement öffnen, ähnlich einer `ComboBox`. In dem geöffneten Formular bzw. Steuerelement werden die Daten, die das Steuerelement verwaltet, in einer anderen Form dargestellt als im Steuerelement selbst. Ein Steuerelement, das der Eingabe einer Zeit dient, zeigt in einem geöffneten Formular z. B. eine (einstellbare) analoge Uhr an.

Wenn Sie so etwas programmieren wollen, müssen Sie die Position des zu öffnenden Formulars bzw. Steuerelements so berechnen, dass dieses an einer idealen Position angezeigt wird. Liegt das Referenz-Steuerelement z. B. nahe dem unteren Rand des Bildschirms, muss das geöffnete Formular/Steuerelement oberhalb des Steuerelements angezeigt werden. Ähnliches gilt für den linken, den rechten und den oberen Rand des Bildschirms. Und dabei sollten Sie auch noch berücksichtigen, dass das System mehrere Bildschirme besitzen kann.

Die Methode `DockToControl` aus Listing 444 nimmt Ihnen diese Arbeit ab. Am ersten Argument übergeben Sie eine Referenz auf das Parent-Formular, am zweiten eine Referenz auf das Steuerelement, nach dem das Steuerelement oder Formular ausgerichtet werden soll, und am letzten Argument das Steuerelement bzw. Formular, dessen Position Sie setzen wollen.

`DockToControl` ermittelt zunächst die auf den Bildschirm bezogene Position des Referenz-Steuerelements. Dabei werden eventuelle Container-Steuerelemente berücksichtigt, auf denen das Steuerelement angelegt ist. Die Position eines Steuerelements bezieht sich ja immer auf seinen Container. Danach geht `DockToControl` alle Bildschirme des Systems durch, um den Bildschirm zu ermitteln, auf dem die X-Position des Referenz-Steuerelements liegt. Wie ich es bereits in Rezept 232 erläutert habe, ist das `Screen`-Array, das die `AllScreens`-Eigenschaft der `Screen`-Klasse verwaltet, nicht nach der X-Position der Bildschirme sortiert. Deswegen wird das Array wieder über den Aufruf von `Array.Sort` mit einem Lambda-Ausdruck sortiert. Das sortierte Array wird nun durchlaufen, um den Bildschirm zu ermitteln, auf dem der linke Rand des Referenz-Steuerelements liegt. Danach folgt ein wenig Rechenarbeit, die die Position des Steuer-

elements so berechnet, dass dieses komplett angezeigt wird. Schließlich wird die berechnete Position zurückgegeben, wobei diese in auf das Parent-Formular bezogene Koordinaten umgerechnet wird, wenn es sich bei dem anzudockenden Objekt um ein Steuerelement handelt.

Zum Kompilieren des Quellcodes in Listing 444 müssen Sie die Namensräume `System`, `System.Windows.Forms`, `System.Drawing` und `System.Collections.Generic` importieren.

```
/* Setzt die optimale Position eines Formulars bezogen
   auf die Position eines Steuerelements */
public static void DockToControl(Form parentForm,
   Control referenceControl, Control control)
{
   // Die auf den Bildschirm bezogene Position des Referenz-Steuerelements
   // ermitteln.
   int xOffset = 0;
   int yOffset = 0;
   Control testControl = referenceControl;
   while ((testControl.Parent is Form) == false)
   {
      xOffset += testControl.Parent.Location.X;
      yOffset += testControl.Parent.Location.Y;
      testControl = testControl.Parent;
   }
   Point p = new Point(referenceControl.Location.X + xOffset,
      referenceControl.Location.Y + yOffset);
   Point controlScreenLocation = parentForm.PointToScreen(p);

   // Ermitteln, auf welchem Bildschirm die X-Position
   // des Steuerelements liegt
   Screen[] screens = (Screen[])Screen.AllScreens.Clone();
   Array.Sort(screens, (x, y) => x.Bounds.Left.CompareTo(y.Bounds.Left));
   Screen controlScreen = screens[0];
   foreach (Screen screen in screens)
   {
      if (controlScreenLocation.X >= screen.Bounds.Left &&
         controlScreenLocation.X <= screen.Bounds.Left + screen.Bounds.Width)
      {
         controlScreen = screen;
         break;
      }
   }

   // Basis-Position setzen
   Point location = new Point(controlScreenLocation.X,
      controlScreenLocation.Y);
```

Listing 444: Methode zum Andocken eines Formulars oder Steuerelements an ein Steuerelement

```
   // Ermitteln, ob das Steuerelement/Formular nach unten noch passt
   if (location.Y + referenceControl.Height + control.Height <=
      controlScreen.WorkingArea.Bottom)
   {
      location.Y += referenceControl.Height + 1;
   }
   else
   {
      location.Y = location.Y - control.Height;
   }

   // Ermitteln, ob das Steuerelement/Formular nach rechts noch passt
   if (location.X + referenceControl.Width + control.Width >
      controlScreen.WorkingArea.Right)
   {
      location.X = controlScreenLocation.X +
         referenceControl.Width - control.Width;
      if (location.X + control.Width > controlScreen.WorkingArea.Right)
      {
         location.X = controlScreen.Bounds.Right - control.Width;
      }
   }

   // Ermitteln, ob das Steuerelement/Formular nach links noch passt
   if (location.X < controlScreen.WorkingArea.Left)
   {
      location.X = controlScreen.WorkingArea.Left;
   }

   // Ermitteln, ob das Steuerelement/Formular nach unten noch passt
   if (location.Y + control.Height > controlScreen.WorkingArea.Bottom)
   {
      location.Y = controlScreen.WorkingArea.Bottom - control.Height;
   }

   // Die Position des Steuerelements bzw. Formular setzen
   if ((control is Form) == false)
   {
      control.Location = parentForm.PointToClient(location);
   }
   else
   {
      control.Location = location;
   }
}
```

Listing 444: Methode zum Andocken eines Formulars oder Steuerelements an ein Steuerelement (Forts.)

Listing 445 zeigt eine Beispielanwendung, bei der eine ListBox geöffnet wird, wenn der Anwender auf einen Schalter neben einer TextBox klickt. Dieses Beispiel zeigt gleich auch, wie Sie ein dynamisch erzeugtes Steuerelement anzeigen und dafür sorgen, dass dieses auch wieder korrekt geschlossen wird. Das Beispiel fügt die dynamisch erzeugte ListBox der Controls-Auflistung des Formulars hinzu und nicht der Controls-Auflistung des Container-Steuerelements (eine GroupBox), auf dem die TextBox angelegt ist. Der Grund dafür ist, dass die ListBox so über allen andern Steuerelementen angezeigt wird.

```
private void openControlButton_Click(object sender, EventArgs e)
{
   // ListBox erzeugen und initialisieren
   ListBox listBox = new ListBox();
   listBox.Leave += new EventHandler(this.listBox_Leave);
   listBox.Click += new EventHandler(this.listBox_Click);
   listBox.KeyPress += new KeyPressEventHandler(this.listBox_KeyPress);
   for (int i = 0; i < 10; i++)
   {
      listBox.Items.Add("Item " + i);
   }
   listBox.SelectedIndex = 0;

   // ListBox an die TextBox docken
   FormUtils.DockToControl(this, this.demoTextBox, listBox);

   // ListBox der Controls-Auflistung des Formulars hinzufügen
   // und anzeigen
   this.Controls.Add(listBox);
   listBox.Show();
   listBox.BringToFront();
   listBox.Focus();
}

private void listBox_Click(object sender, EventArgs e)
{
   // Den aktuellen Eintrag übernehmen und die ListBox
   // wieder schließen
   this.demoTextBox.Text = (((ListBox)sender).SelectedItem.ToString());
   this.Controls.Remove((ListBox)sender);
   ((ListBox)sender).Hide();
}

private void listBox_KeyPress(object sender, KeyPressEventArgs e)
{
   if (e.KeyChar == (char)Keys.Enter)
   {
```

Listing 445: Beispielanwendung der DockToControl-Methode

```
      // Den aktuellen Eintrag übernehmen und die ListBox
      // wieder schließen
      this.demoTextBox.Text = (((ListBox)sender).SelectedItem.ToString());
      this.Controls.Remove((ListBox)sender);
      ((ListBox)sender).Hide();
   }
}

private void listBox_Leave(object sender, EventArgs e)
{
   // Die ListBox schließen
   this.Controls.Remove((ListBox)sender);
   ((ListBox)sender).Hide();
}
```

Listing 445: Beispielanwendung der DockToControl-Methode (Forts.)

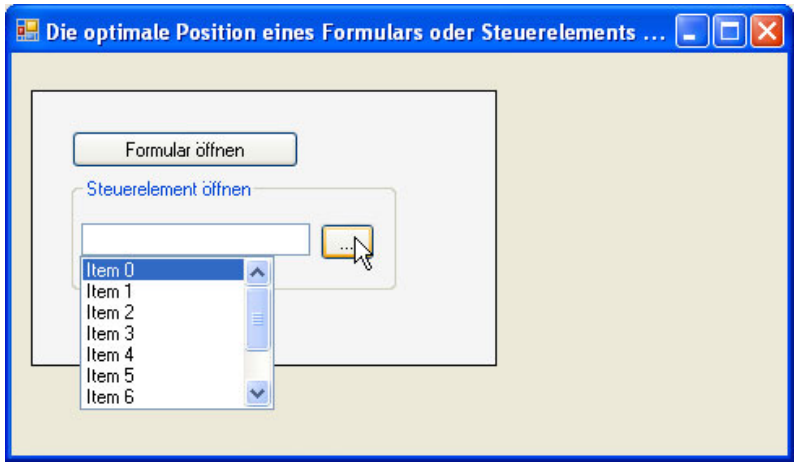


Abbildung 132: Die Beispielanwendung hat eine ListBox geöffnet, die an eine TextBox angedockt ist

236 Das Hauptformular einer Anwendung ermitteln

Wenn Sie (z. B. in einer Klassenbibliothek) das Hauptformular der Anwendung ermitteln müssen, die die Klassenbibliothek verwendet, können Sie dazu den Windows-Handle des Hauptfensters verwenden, den Sie aus der Eigenschaft `MainWindowHandle` des aktuellen Prozesses auslesen. Über `Form.FromHandle` erhalten Sie eine Referenz auf das dem Handle entsprechende (Haupt-)Formular:

```
Form mainForm = (Form)Form.FromHandle(
   Process.GetCurrentProcess().MainWindowHandle);
```

Listing 446: Ermitteln des Hauptformulars einer Anwendung

Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume `System.Windows.Forms` und `System.Diagnostics` einbinden.

237 Menüs in der Laufzeit erstellen oder erweitern

Viele Anwendungen erfordern, dass Menüs in der Laufzeit erstellt oder erweitert werden. Ein gutes Beispiel dafür sind die Datei-Menüs von Office-Anwendungen, die beim Start der Anwendung und während der Arbeit automatisch die letzten geöffneten Dateien zum Öffnen anbieten. Das Erstellen eines Laufzeitmenüs ist nun sehr einfach, weil ein `MenuItem`-Objekt (das ein Menü oder einen Menüeintrag darstellt) eine Auflistung `MenuItems` besitzt, über dessen `Add`-Methode Sie neue `MenuItem`-Instanzen hinzufügen können. Das einzige Problem (und damit der Grund für dieses Rezept) ist, dass Sie Menüeinträgen, die Befehle darstellen, einen Delegaten für die Behandlung des `Click`-Ereignisses zuweisen und in diesem den betätigten Menüeintrag ermitteln müssen.

Als Beispiel verwende ich ein Formular mit einem einfachen Menü (Abbildung 133).

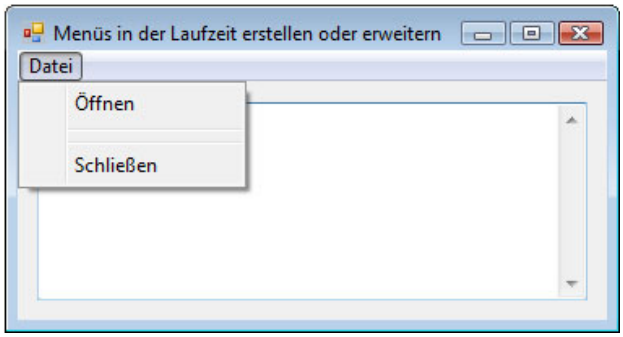


Abbildung 133: Das Formular, dessen Datei-Menü in der Laufzeit erweitert werden soll

Im `Load`-Ereignis soll das Datei-Menü um die Namen der Textdateien erweitert werden, die zuletzt geöffnet wurden. Ein Klick auf diese neuen Menüeinträge soll die jeweils dahinter stehende Datei öffnen und deren Inhalt in die `TextBox` laden. Dazu ist zunächst eine Methode für das `Click`-Ereignis des Menüeintrags notwendig:

```
private void DynamicFileMenu_Click(object sender, System.EventArgs e)
{
   // Ermitteln des Dateinamens
   MenuItem menuItem = (MenuItem)sender;
   string filename = menuItem.Text;

   // Einlesen der Datei
   StreamReader sr = null;
   try
   {
      sr = new StreamReader(filename, Encoding.Default);
```

Listing 447: Methode für das Click-Ereignis der dynamisch erzeugten Menüeinträge

```
      this.fileContentTextBox.Text = sr.ReadToEnd();
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   finally
   {
      try
      {
         sr.Close();
      }
      catch { }
   }
}
```

Listing 447: Methode für das Click-Ereignis der dynamisch erzeugten Menüeinträge (Forts.)

Diese Methode kann den Menüeintrag einfach über dessen Text identifizieren, was in Listing 448 auch so umgesetzt ist.

In der Delegat-Methode für das `Load`-Ereignis des Formulars wird das Datei-Menü nun dynamisch erweitert:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
   // Hinzufügen von Menübefehlen für alle Textdateien
   // im Programm-Verzeichnis
   DirectoryInfo programDirectory =
      new DirectoryInfo(Application.StartupPath);
   FileInfo[] files = programDirectory.GetFiles("*.txt");
   for (int i = 0; i < files.Length && i < 9; i++)
   {
      // MenuItem-Objekt erzeugen und den Text und die Click-Ereignismethode
      // übergeben
      MenuItem menuItem = new MenuItem(files[i].Name,
         new EventHandler(this.DynamicFileMenu_Click));

      // Menü ab dem Index 2 + i an das Dateimenü anfügen
      this.fileMenu.MenuItems.Add(2 + i, menuItem);
   }
}
```

Listing 448: Dynamisches Erweitern eines Menüs

Das Beispiel geht alle Textdateien im Anwendungs-Ordner durch und fügt deren Namen an das Dateimenü an. Beim Erzeugen des `MenuItem`-Objekts werden der Text des Eintrags und ein

Delegat für die `Click`-Behandlungsmethode übergeben. Am im Beispiel nicht verwendeten dritten Argument können Sie übrigens noch einen Shortcut für das Menü definieren. Der Menüeintrag wird dann dem Dateimenü (das über das Feld `fileMenu` angesprochen wird) über dessen `Add`-Methode angefügt. Wie Sie sehen, gehen Sie beim Ändern eines Menüs mit einer ganz normalen Auflistung um. Über die `Remove`-Methode können Sie Menüeinträge z. B. auch wieder löschen, was beim dynamischen Verändern des Menüs während der Programmausführung notwendig wird.

Das Ergebnis zeigt Abbildung 134.

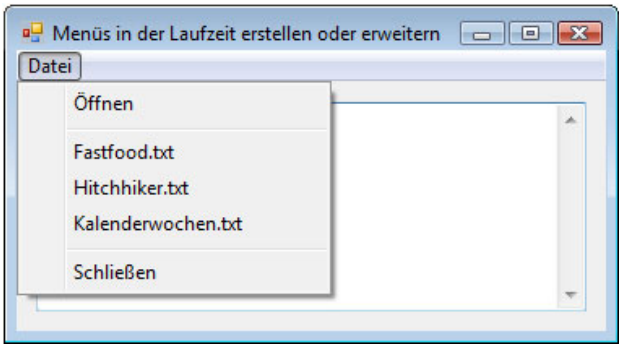


Abbildung 134: Das Programm mit dem dynamisch erweiterten Datei-Menü

238 Steuerelemente mit transparentem Hintergrund

Eine einfach gelöste, aber in Newsgroups immer wieder auftauchende Frage ist die, wie erreicht wird, dass Steuerelemente einen transparenten Hintergrund besitzen. Das gilt besonders für `Label`, die ihre Hintergrundfarbe zwar per Voreinstellung an die Hintergrundfarbe des Formulars anpassen, bei Formularen, denen ein Bild zugewiesen ist, aber sehr unschön aussehen.

Die Lösung dieses Problems ist einfach: Weisen Sie der Hintergrundfarbe des Steuerelements die Farbe `Color.Transparent` zu. Im Eigenschaftenfenster von Visual Studio 2008 finden Sie diese Farbe im Register WEB.

Hinweis

Nicht alle Steuerelemente unterstützen transparente Hintergrundfarben. Einige werfen beim Versuch, die transparente Farbe zu verwenden, eine Exception. Lösen können Sie dieses Problem, indem Sie ein eigenes Steuerelement von diesem Steuerelement ableiten und im Konstruktor transparente Farben erlauben:

```
this.SetStyle(ControlStyles.SupportsTransparentBackColor, true);
```

Ebenfalls interessant sind halb-durchsichtige Steuerelemente. Dies erreichen Sie, indem Sie im Programm eine Farbe zuweisen, deren Alpha-Wert kleiner als 255 ist. Der Alpha-Wert bestimmt die Transparenz. Ein Wert von 255 führt zu einer Undurchsichtigkeit, 0 ist vollständig transparent. Dazwischen können Sie alle Werte einsetzen, um die Transparenz gezielt einzustellen. Dazu verwenden Sie die Methode `FromArgb` der `Color`-Struktur (aus dem Namens-

raum System.Drawing) in der Variante mit vier Argumenten und geben den Alpha-Wert am ersten Argument an. So können Sie z. B. ein halb-transparent-hell-stahlblaues Label erzeugen:

```
Color color = Color.FromArgb(100, Color.LightSteelBlue.R,
   Color.LightSteelBlue.G, Color.LightSteelBlue.B);
this.demoLabel3.BackColor = color;
```

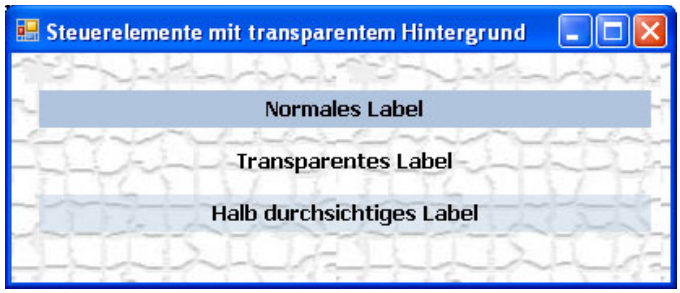


Abbildung 135: Undurchsichtiges, transparentes und halb-transparentes Label auf einem Formular

239 TextBox automatisch beim Eintritt selektieren

Die Selektion von TextBox-Steuerelementen wird beim Öffnen eines Formulars automatisch so eingestellt, dass der gesamte Text beim Fokuserhalt selektiert ist. TextBoxen merken sich die aktuelle Selektion, sodass diese wiederhergestellt wird, wenn der Fokus von der TextBox wegbewegt und wieder auf diese gesetzt wird. Hat der Anwender die Selektion zwischenzeitlich geändert, wird die TextBox beim nächsten Fokuserhalt nicht mehr vollständig selektiert.

Wenn Sie eine sichere vollständige Selektion beim Fokuserhalt erreichen wollen, können Sie im Enter-Ereignis der TextBox einfach die SelectAll-Methode aufrufen:

```
private void firstNameTextBox_Enter(object sender, System.EventArgs e)
{
   this.firstNameTextBox.SelectAll();
}
```

Listing 449: Vollständiges Selektieren des Inhalts einer TextBox beim Fokuserhalt

Da dies recht aufwändig ist, wäre es vorteilhaft, eine eigene TextBox zu entwickeln, die diese Arbeit automatisch erledigt. Dazu verwenden Sie in Visual Studio 2008 idealerweise ein neues Projekt vom Typ *Windows-Steuerelementbibliothek*. Die Klasse könnte dann aussehen wie die Klasse AutoSelectTextBox in Listing 450. Diese Klasse ist von TextBox abgeleitet und implementiert eine Eigenschaft AutoSelect, die das automatische Selektieren steuert. Die Attribute Category und Description sind übrigens nur enthalten, damit das neue Steuerelement innerhalb des Eigenschaftenfensters von Visual Studio 2008 in der Kategorie *Verhalten* einsortiert wird und eine Beschreibung zur neuen Eigenschaft erscheint.

Zur Umsetzung der automatischen Selektion überschreibt die Klasse einfach die geschützte Methode `OnEnter`.

```
public class AutoSelectTextBox : TextBox
{
    /* Gibt an, ob der Inhalt der TextBox beim Fokuserhalt automatisch
       selektiert wird */
    [Category("Behavior")]
    [Description("Gibt an, ob der Inhalt der TextBox beim Fokuserhalt " +
       "automatisch selektiert wird")]
    public bool AutoSelect
    {
        get;
        set;
    }

    /* OnEnter wird überschrieben um den gesamten Inhalt zu selektieren
       wenn die TextBox den Fokus erhält und AutoSelect true ist. */
    protected override void OnEnter(EventArgs e)
    {
        if (this.AutoSelect)
        {
            this.SelectAll();
        }
    }
}
```

Listing 450: Klasse für eine erweiterte TextBox, die das automatische Selektieren beim Fokuserhalt ermöglicht

Hinweis

Nur für den Fall, dass Sie nicht wissen, wie Sie dieses neue Steuerelement in Visual Studio 2008 einsetzen: Zum einen können Sie die Klasse einfach in ein Projekt integrieren. Visual Studio 2008 zeigt alle Steuerelemente der aktuellen Projektmappe automatisch in separaten (neuen) Registern in der Toolbox an und Sie können diese wie normale Steuerelemente verwenden.

Alternativ können Sie eine Klassenbibliothek erzeugen und die `AutoSelectTextBox`-Klasse darin integrieren. Wenn Sie dieses Projekt einer Projektmappe hinzufügen, die bereits ein Projekt für eine Windows-Anwendung enthält (was während der Entwicklung von Steuerelementen sehr hilfreich ist), können Sie die Klassenbibliothek (über das Projekt) in der Windows-Anwendung direkt referenzieren. Auch in diesem Fall fügt Visual Studio 2008 für die in dem Klassenbibliotheks-Projekt enthaltenen Steuerelemente der Toolbox ein Register an.

Falls Sie die Klassenbibliothek kompilieren und die kompilierte Assembly in Windows-Anwendung-Projekten referenzieren wollen, müssen Sie die Steuerelemente der Assembly allerdings der Toolbox von Hand hinzufügen. Dazu wählen Sie im Kontextmenü der Toolbox den Befehl TOOLBOX ANPASSEN. Suchen Sie die Assembly des Steuerelements und fügen Sie diese hinzu. Nun können Sie das Steuerelement verwenden wie jedes andere auch.

240 TextBox auf Zahleingaben beschränken

In meinen Seminaren werde ich immer wieder gefragt, wie man erreichen kann, dass eine `TextBox` nur Zahleingaben zulässt. Das .NET Framework bietet dazu leider keine direkten Möglichkeiten. Sie können natürlich spezielle `TextBox`-Steuerelemente über Drittthersteller oder aus dem Internet beziehen. Sie können aber auch selbst eine solche `TextBox` programmieren. Da Sie dabei beachten müssen, dass Eingaben über die Tastatur, das Einfügen über die Zwischenablage und das Schreiben in die `Text`-Eigenschaft kontrolliert werden müssen, sollten Sie dazu direkt ein eigenes Steuerelement erstellen. Ansonsten haben Sie bei der Programmierung eines Formulars mit einigen `TextBox`-Steuerelementen einfach zu viel Arbeit.

Meine aus einem Artikel bei *The Code Project* (*www.codeproject.com/cs/miscctrl/NumEdit.asp*) abgeleitete Klasse wird also von `TextBox` abgeleitet.

Zum Kompilieren des Quellcodes müssen Sie die Namensräume `System`, `System.ComponentModel` und `System.Windows.Forms` importieren.

Um den Eingabetyp bestimmen zu können, habe ich zunächst eine Aufzählung `InputTypes` implementiert. Da die Klasse `NumberTextBox` bei ungültigen Eingaben ein Ereignis aufrufen soll, habe ich außerdem die dazu notwendige Ereignisargumentklasse implementiert:

```
/* Aufzählung für den Typ einer numerischen Eingabe */
public enum NumberInputTypes
{
    /* Als Eingabe sind Zahlen mit Nachkommastellen möglich */
    Double,

    /* Als Eingabe sind Zahlen ohne Nachkommastellen möglich */
    Integer
}

/*  Ereignisargumentklasse für das NumberTextBox.InvalidInput-Ereignis */
public class InvalidInputEventArgs : EventArgs
{
    /* Die ungültige Eingabe */
    public string Input
    {
```

Listing 451: Aufzählung für den Typ einer numerischen Eingabe und Ereignisargument-Klasse für das InvalidInput-Ereignis

```
        private set;
        get;
    }

    /* Konstruktor */
    internal InvalidInputEventArgs(string input)
    {
        this.Input = input;
    }
}
```

Listing 451: Aufzählung für den Typ einer numerischen Eingabe und Ereignisargument-Klasse für das InvalidInput-Ereignis (Forts.)

Die Klasse `NumberTextBox` beginnt mit der Deklaration der Eigenschaft `InputType`, die die Art der Eingabe steuert, der Deklaration des Ereignisses `InvalidInput`, das aufgerufen wird, wenn ungültige Daten eingegeben werden, und der Methode `OnInvalidInput`. `OnInvalidInput` entspricht der Microsoft-Konvention für Ereignisse und macht nichts anderes, als das `InvalidInput`-Ereignis aufzurufen.


```
public class NumberTextBox : TextBox
{
    /* Gibt an, welche Eingaben die TextBox zulässt */
    [Category("Behavior")]
    [Description("Gibt an, welche Eingaben die TextBox zulässt")]
    public NumberInputTypes InputType
    {
        get;
        set;
    }

    /* Wird aufgerufen, wenn der Anwender ungültige Daten eingibt */
    public event EventHandler<InvalidInputEventArgs> InvalidInput;

    /* Ruft das <see cref="InvalidInput"/>-Ereignis auf */
    protected virtual void OnInvalidInput(string input)
    {
        if (this.InvalidInput != null)
        {
            this.InvalidInput(this, new InvalidInputEventArgs(input));
        }
    }
```

Listing 452: Beginn der Klasse NumberTextBox

Nun folgt die Methode `CheckInput`, die überprüft, ob die Eingabe in Ordnung ist. Diese Methode bekommt den potenziell neuen Text der `TextBox` übergeben und überprüft diesen einfach über eine Konvertierung nach `double` bzw. `long` auf Gültigkeit. Im Fehlerfall ruft `CheckInput` über `OnInvalidInput` das Ereignis `InvalidInput` auf und gibt `false` zurück:

```
private bool CheckInput(string input)
{
   switch (this.inputType)
   {
      case InputTypes.Double:
         try
         {
            Convert.ToDouble(input);
            return true;
         }
         catch
         {
            this.OnInvalidInput(input);
            return false;
         }

      default:  // InputTypes.Integer
         try
         {
            Convert.ToInt64(input);
            return true;
         }
         catch
         {
            this.OnInvalidInput(input);
            return false;
         }
   }
}
```

Listing 453: Methode zum Überprüfen, ob die Eingabe in Ordnung ist

Um ein Schreiben in die `Text`-Eigenschaft zu überprüfen, wird diese überschrieben. Im `set`-Accessor ruft `Text CheckInput` auf. Nur wenn der Text in Ordnung ist, wird dieser über die geerbte `Text`-Eigenschaft in die `TextBox` geschrieben:

```
public override string Text
{
   get
   {
```

Listing 454: Überschreiben der Text-Eigenschaft

```
            return base.Text;
        }

        set
        {
            if (this.CheckInput(value))
            {
                base.Text = value;
            }
        }
    }
```

Listing 454: Überschreiben der Text-Eigenschaft (Forts.)

Die `OnKeyPress`-Methode wird ebenfalls überschrieben, um Tastatureingaben überprüfen zu können. Dabei wird das eingegebene Zeichen so in den alten Text integriert, dass eine eventuelle Selektion oder eine Cursorposition vor dem Ende des Textes berücksichtigt wird. Der potenziell neue Text wird dann wieder über `CheckInput` überprüft. Im Fehlerfall setzt `OnKeyPress` `e.Handled` auf `true`, damit die Eingabe nicht weiter verarbeitet wird. Die Überprüfung erfolgt nur dann, wenn es sich nicht um ein Steuerzeichen wie z. B. einen Tabulator oder die Backspace-Taste handelt.


```
protected override void OnKeyPress(KeyPressEventArgs e)
{
    if (Char.IsControl(e.KeyChar) == false)
    {
        // Den potenziell neuen Text zusammensetzen
        string newText = base.Text.Substring(0, base.SelectionStart)
            + e.KeyChar.ToString() + base.Text.Substring(
            base.SelectionStart + base.SelectionLength);

        if (this.CheckInput(newText) == false)
        {
            // Die Eingabe führt zu einem ungültigen Ergebnis,
            // also verwerfen
            e.Handled = true;
        }
    }

    base.OnKeyPress(e);
}
```

Listing 455: Überschreiben der OnKeyPress-Methode zum Überprüfen von Tastatureingaben

Schließlich müssen noch die Daten überprüft werden, die der Anwender aus der Zwischenablage einfügt. Dazu wird die `WndProc`-Methode überschrieben, die immer dann aufgerufen wird, wenn Windows oder ein Programm eine Nachricht an das Fenster sendet. Die Nachricht,

die uns hier interessiert, ist WM_PASTE (0x0302). WM_PASTE wird immer dann gesendet, wenn der Anwender versucht, Daten über die Zwischenablage in die TextBox einzufügen. Ich nutze diese Nachricht, um den Inhalt der Zwischenablage auszulesen und über CheckInput zu überprüfen, ob die neue Eingabe gültig wäre. Dabei wird berücksichtigt, dass in der TextBox gerade Text selektiert sein kann. Der einzufügende Text wird vor der Überprüfung an die richtige Stelle im aktuellen Text eingefügt. Ist der Inhalt nicht gültig, wird die geerbte WndProc-Methode nicht aufgerufen und damit ein Einfügen verhindert:

```
protected override void WndProc(ref Message m)
{
   const int WM_PASTE = 0x0302;

   switch (m.Msg)
   {
      case WM_PASTE:
         IDataObject dataObject = Clipboard.GetDataObject();
         if (dataObject.GetDataPresent(DataFormats.Text))
         {
            // Text aus der Zwischenablage auslesen
            string clipboardText =
               dataObject.GetData(DataFormats.Text).ToString();

            // Die potenziell neue Eingabe zusammensetzen
            string input = this.Text;
            if (this.SelectionLength > 0)
            {
               // Wenn gerade Text selektiert ist, wird der Inhalt
               // der Zwischenablage in die Selektion eingefügt
               if (this.SelectionStart == 0)
               {
                  input = clipboardText +
                     input.Substring(this.SelectionLength,
                     input.Length - this.SelectionLength);
               }
               else
               {
                  input = input.Substring(0, this.SelectionStart) +
                     clipboardText + input.Substring(this.SelectionStart +
                     this.SelectionLength, input.Length -
                     this.SelectionStart - this.SelectionLength);
               }
            }
            else
            {
```

Listing 456: Die überschriebene WndProc-Methode zur Verhinderung des Einfügens ungültiger Daten über die Zwischenablage

```
                // Wenn kein Text selektiert ist, wird der Inhalt der
                // Zwischenablage an der Cursorposition eingefügt
                input = input.Substring(0, this.SelectionStart) +
                   clipboardText + input.Substring(this.SelectionStart,
                   input.Length - this.SelectionStart);
             }

             if (this.CheckInput(input) == true)
             {
                // Die geerbte Methode aufrufen um den Text einzufügen
                base.WndProc(ref m);
             }
             else
             {
                // Hier wird nichts gemacht, damit der
                // Text nicht eingefügt wird
             }
          }
          else
          {
             // Die geerbte Methode aufrufen um den Text einzufügen
             base.WndProc(ref m);
          }
          break;

       default:
          // Die geerbte Methode aufrufen
          base.WndProc(ref m);
          break;
    }
}
```

Listing 456: Die überschriebene WndProc-Methode zur Verhinderung des Einfügens ungültiger Daten über die Zwischenablage (Forts.)

241 Auf Return in einer TextBox reagieren

Eine Reaktion auf die Betätigung der Return-Taste in einer `TextBox` ist eigentlich viel zu einfach, um hier als Rezept zu erscheinen. Dazu überprüfen Sie im `KeyPress`-Ereignis lediglich die betätigte Taste auf '\r'. Vielleicht wissen Sie aber nicht, wie Sie den Systemton abschalten können, den Windows automatisch bei einem Return in einer `TextBox` ausgibt ☺.

```
private void demoTextBox_KeyPress(object sender, KeyPressEventArgs e)
{
   if (e.KeyChar == '\r')
```

Listing 457: Reaktion auf die Betätigung der Return-Taste in einer TextBox

```
    {
        // Ereignis als behandelt kennzeichnen um den Systemton zu verhindern
        e.Handled = true;

        // Weitere Programmierung
    }
}
```

Listing 457: Reaktion auf die Betätigung der Return-Taste in einer TextBox (Forts.)

242 Bei der Betätigung der Return-Taste die Tab-Taste simulieren

Viele Anwender wünschen, dass der Eingabecursor in einer Anwendung mit der Return-Taste auf das nächste Eingabesteuerelement bewegt werden kann. Obwohl dieses Verhalten nicht Windows-konform ist, macht es bei vielen Anwendungen dennoch Sinn. Dies gilt besonders dann, wenn Anwender die Dateneingabe in einem Formular für jeweils neue Vorgänge immer wieder wiederholen müssen. Aber ich will nicht weiter über den Sinn dieses Wunsches diskutieren, sondern eine Lösung anbieten.

Prinzipiell können Sie natürlich einfach die Return-Taste in den `KeyPress`-Ereignissen der Steuerelemente abfangen und den Fokus auf das nächste Steuerelement setzen. In meinen Augen macht das aber viel zu viel Arbeit und ist auch mit Nacharbeit verbunden, wenn neue Steuerelemente hinzukommen. Besser wäre es, Steuerelemente zu besitzen, die bei der Betätigung der Return-Taste die Tab-Taste simulieren. Sie sollten also für die benötigten Steuerelemente neue Klassen erzeugen, die von den Klassen der benötigten Steuerelemente abgeleitet werden. In der überschriebenen `OnKeyPress`- oder der `OnKeyDown`-Methode können Sie auf die Betätigung der Return-Taste reagieren. Leider können Sie aber nicht einfach den Tastencode verändern um das Tabulator-Problem zu lösen. In der `OnKeyDown`-Methode ist die `KeyCode`-Eigenschaft des Ereignisargument-Objekts schreibgeschützt. Die Ereignisargument-Eigenschaft `KeyChar` der `OnKeyPress`-Methode erlaubt zwar die Veränderung der betätigten Taste, aber das Schreiben des Tabulator-Zeichens ('\t') in diese Eigenschaft führt zu keinem Ergebnis. Der Grund dafür ist, dass die Tabulator-Taste nicht von dem Steuerelement, sondern von seinem Container behandelt wird.

Die Lösung dieses Problems ist, dass Sie den Container des Steuerelements auffordern, die Tab-Taste zu verarbeiten. Die `ContainerControl`-Klasse besitzt dazu die Methode `ProcessTabKey`, die genau das macht. Leider ist diese Methode nicht öffentlich, sondern `protected` deklariert. Aber das sollte kein Hindernis sein ☺. Sie können nämlich über Reflection alle Methoden eines Objekts aufrufen, auch private und geschützte.

Listing 458 zeigt am Beispiel einer `TextBox`, wie es geht. Diese `Return2TabTextBox` implementiert zunächst die Eigenschaft `SimulateTabOnReturn`, die das Verhalten bei der Betätigung der Return-Taste steuert. In der überschriebenen `OnKeyPress`-Methode wird diese Eigenschaft abgefragt um bei der Betätigung der Return-Taste die Tab-Taste zu simulieren. Dazu ermittelt diese Methode zunächst das Container-Steuerelement. Normalerweise könnte dazu einfach die `Parent`-Eigenschaft des Steuerelements abgefragt werden. Diese ist aber vom Typ `Control` und eben nicht vom Typ `ContainerControl`. Die Klasse `Control` besitzt keine `ProcessTabKey`-

Methode. Normalerweise könnte die Parent-Eigenschaft nach ContainerControl gecastet werden, aber irgendeinen Sinn muss es haben, dass diese Eigenschaft nicht vom Typ ContainerControl ist. Deshalb geht das Programm den sicheren Weg und ermittelt das Container-Steuerelement über die rekursive private Methode GetContainerControl. Konnte ein Container ermittelt werden, ruft OnKeyPress die ProcessTabKey-Methode über die InvokeMember-Methode des Typs von ContainerControl auf, wobei dem Argument forward dieser Methode true übergeben wird. Falls Sie sich fragen, woher ich über diese Methode so genau Bescheid weiß: Dazu habe ich den in der Einführung erwähnten Reflector von Lutz Roeder verwendet ☺.

Die Klasse Return2TabTextBox erfordert den Import der Namensräume System, System.Windows.Forms und System.ComponentModel.

```
/* TextBox, die es erlaubt, bei der Betätigung von Return die
   TAB-Taste zu simulieren */
public class Return2TabTextBox : TextBox
{
   private bool simulateTabOnReturn = true;
   /* Gibt an, ob bei der Betätigung der Return-Taste
      die Tab-Taste simuliert wird */
   [DefaultValue(true)]
   [Category("Behavior")]
   public bool SimulateTabOnReturn
   {
      get { return this.simulateTabOnReturn; }
      set { this.simulateTabOnReturn = value; }
   }

   /* OnKeyPress wird überschrieben, um bei der Betätigung der
      Return-Taste die TAB-Taste zu simulieren */
   protected override void OnKeyPress(KeyPressEventArgs e)
   {
      if (this.simulateTabOnReturn && e.KeyChar == '\r')
      {
         // Return wurde betätigt und soll in Tab umgewandelt werden
         ContainerControl container = this.GetContainerControl(this);
         if (container != null)
         {
            // Die leider geschützte ProcessTabKey-Methode
            // des ContainerControls über Reflection aufrufen
            Type type = container.GetType();
            type.InvokeMember("ProcessTabKey", BindingFlags.InvokeMethod |
               BindingFlags.NonPublic | BindingFlags.Instance, null,

               container, new object[] { true });
```

Listing 458: TextBox, die eine Betätigung der Return-Taste in eine Betätigung der Tab-Taste umsetzt

```
            // Das Ereignis als behandelt kennzeichnen
            e.Handled = true;
         }
      }

      // Die geerbte Methode aufrufen
      base.OnKeyPress(e);
   }

   /* Liefert das ContainerControl eines Steuerelements  */
   private ContainerControl GetContainerControl(Control control)
   {
      if (control.Parent != null)
      {
         if (control.Parent is ContainerControl)
         {
            return (ContainerControl)control.Parent;
         }
         else
         {
            // Rekursiv aufrufen um den Parent des
            // übergebenen Steuerelements zu überprüfen
            return this.GetContainerControl(control.Parent);
         }
      }
      else
      {
         return null;
      }
   }
}
```

Listing 458: TextBox, die eine Betätigung der Return-Taste in eine Betätigung der Tab-Taste umsetzt (Forts.)

243 Die Position des Eingabecursors in einer TextBox, RichTextBox oder MaskedTextBox ermitteln

Gute Texteditoren zeigen die aktuelle Position des Eingabecursors meist in der Statuszeile an. Wollen Sie diese Position in Ihren Programmen ebenfalls anzeigen oder für spezielle Programmierungen (wie z. B. das Markieren der aktuellen Zeile) einsetzen, erfahren Sie diese leider nicht über Eigenschaften oder Methoden der `TextBox`-, `RichTextBox`- oder `MaskedTextBox`-Klasse. Zur Ermittlung der aktuellen Position müssen Sie die Windows-API-Funktion `SendMessage` aufrufen. Wie ich bereits in der Einführung zu diesem Buch erläutert habe, sendet `SendMessage` eine Windows-Nachricht an ein Fenster, in unserem Fall an das Fenster der `TextBox`. Senden Sie die Nachricht `EM_LINEFROMCHAR` (`0xC9`), erhalten Sie den (nullbasierten) Index der Zeile zurück, in der sich der Cursor gerade befindet. Über `EM_LINEINDEX` (`0xBB`) erhalten Sie die

Startposition der aktuellen Zeile bezogen auf das erste Zeichen in der TextBox. Mit diesen Informationen können Sie sehr einfach den Index des Cursors bezogen auf den Zeilenanfang berechnen.

So, genug Theorie. Hier ist die Lösung: Die Methode GetTextBoxCursorPosition in Listing 459 berechnet auf die beschriebene Weise den Index und die Startposition der aktuellen Zeile und den aktuellen Spaltenindex. Damit die Methode für alle Textboxen verwendet werden kann, ist das einzige Argument vom Typ TextBoxBase (TextBox-, RichTextBox oder MaskedTextBox sind von dieser Klasse abgeleitet). Die ermittelten Informationen werden in eine Instanz der Struktur TextBoxCursorPosition geschrieben, die schließlich zurückgegeben wird. Die einzelnen Eigenschaften dieser Struktur habe ich so implementiert, dass diese nur innerhalb der (Klassenbibliotheks-)Assembly geschrieben, aber von außen natürlich gelesen werden können.

Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume System, System.Windows.Forms und System.Runtime.InteropServices importieren.

```
/* Deklaration der benötigten API-Funktion */
[DllImport("User32.Dll")]
private static extern int SendMessage(IntPtr hWnd, int msg,
     int wParam, int lParam);

/* Verwaltet Informationen zur Cursorposition in einer TextBox */
public struct TextBoxCursorPosition
{
   /* Der Index der aktuellen Zeile */
   public int RowIndex
   {
      get;
      internal set;
   }

   /* Der Index des Zeichens in der aktuellen Zeile  */
   public int ColumnIndex
   {
      get;
      internal set;
   }

   /* Der Start-Index der aktuellen Zeile */
   public int RowStartIndex
   {
      get;
      internal set;
   }
}
```

Listing 459: Methode zur Ermittlung der aktuellen Cursorposition in einer TextBox

```
/* Liefert Angaben zur Cursorposition in einer von TextBoxBase
   abgeleiteten Klasse  */
public static TextBoxCursorPosition GetTextBoxCursorPosition(
   TextBoxBase textBox)
{
   const int EM_LINEINDEX = 0xBB;
   const int EM_LINEFROMCHAR = 0xC9;

   TextBoxCursorPosition textBoxCursorPosition =
         new TextBoxCursorPosition();

   // Index der aktuellen Zeile ermitteln
   textBoxCursorPosition.RowIndex = SendMessage(textBox.Handle,
         EM_LINEFROMCHAR, -1, 0);

   // Start-Position der aktuellen Zeile ermitteln
   textBoxCursorPosition.RowStartIndex = SendMessage(textBox.Handle,
         EM_LINEINDEX, -1, 0);

   // Index der aktuellen Spalte berechnen
   textBoxCursorPosition.ColumnIndex = textBox.SelectionStart -
         textBoxCursorPosition.RowStartIndex;

   // Ergebnis zurückgeben
   return textBoxCursorPosition;
}
```

Listing 459: Methode zur Ermittlung der aktuellen Cursorposition in einer TextBox (Forts.)

Hinweis

Bei der Verwendung dieser Methode sollten Sie beachten, dass Sie immer die Position bezogen auf den aktuell angezeigten Text erhalten. Ist die Eigenschaft `WordWrap` der `TextBox` auf `true` eingestellt, bricht die `TextBox` die Zeilen am rechten Rand um. Aus einer tatsächlichen Zeile werden dann mehrere dargestellte Zeilen. Wird z. B. eine physikalische Zeile in drei angezeigte Zeilen umbrochen, und der Cursor steht in der dritten Zeile, gibt `GetTextBoxCursorPosition` in der Eigenschaft `TextBoxCursorPosition.RowIndex` den Wert 2 zurück. Wenn die Angabe des Zeilen- und des Spaltenindex mit den tatsächlich vorhandenen Daten übereinstimmen soll, müssen Sie `WordWrap` auf `false` einstellen.

In einem Programm sollten Sie in Methoden für die TextBox-Ereignisse KeyUp, MouseUp und, falls WordWrap true ist, auch in der Methode für Resize einen Aufruf einer privaten Methode implementieren, die die Aktualisierung der Positions-Info vornimmt. Listing 460 aktualisiert auf diese Weise ein Label.

```
private void demoTextBox_KeyUp(object sender, KeyEventArgs e)
{
   this.updatePositionInfo();
}

private void demoTextBox_MouseUp(object sender, MouseEventArgs e)
{
   this.updatePositionInfo();
}

private void demoTextBox_Resize(object sender, EventArgs e)
{
   this.updatePositionInfo();
}

private void updatePositionInfo()
{
   TextBoxCursorPosition textBoxCursorPosition =
      GetTextBoxCursorPosition(this.demoTextBox);
   this.positionInfoLabel.Text =
      "Zeilenindex: " + textBoxCursorPosition.RowIndex +
      ", Spaltenindex: " + textBoxCursorPosition.ColumnIndex +
      ", Startindex der Zeile: " + textBoxCursorPosition.RowStartIndex;
}
```

Listing 460: Beispielhafte Anwendung der GetTextBoxCursorPosition-Methode

Abbildung 136 zeigt ein Beispielprogramm mit einer TextBox, deren WordWrap-Eigenschaft auf true steht.

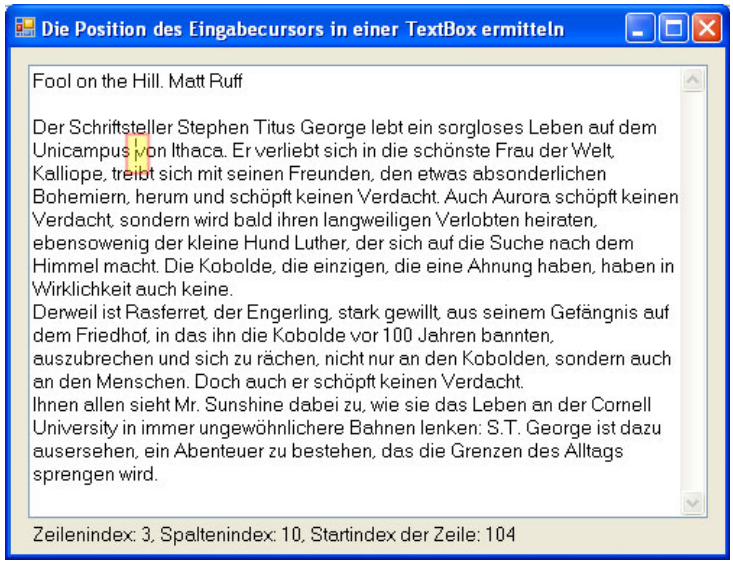


Abbildung 136: Ein Beispielprogramm nutzt GetTextBoxCursorPosition zur Ermittlung der Eingabecursorposition

244 Die angezeigten Zeilen einer MultiLine-TextBox auslesen

Eine `TextBox` mit `MultiLine = true` und `WordWrap = true` bricht den enthaltenen Text automatisch am rechten Rand weich um wenn dieser den Rand erreicht. Wenn Sie den Inhalt der `TextBox` in der Form verarbeiten wollen, in der er angezeigt wird, hilft Ihnen die `TextBox`-Klasse selbst nicht weiter. Die `Lines`-Eigenschaft liefert lediglich die Zeilen, die durch einen harten Zeilenumbruch (CR/LF) getrennt wurden. Um die Zeilen so auszulesen, wie diese angezeigt werden, müssen Sie das Windows-API bemühen.

Die Methode `GetDisplayedTextBoxRows` in Listing 461 setzt zum Lesen der Zeilen die API-Funktion `SendMessage` ein. Sie ermittelt zunächst über die Nachricht `EM_GETLINECOUNT` die Anzahl der Zeilen in der übergebenen `TextBox`. In einer Schleife, in der die Methode die einzelnen Zeilen durchgeht, wird zunächst die Position des ersten Zeichens der Zeile ermittelt, um dann die Länge der Zeile auszulesen. Danach folgt das Einlesen der Zeile, sofern diese zumindest ein Zeichen enthält. Dazu reserviert `GetDisplayedTextBoxRows` zunächst Speicher im nicht verwalteten Bereich, in den `SendMessage` später die einzelnen Bytes der Zeile schreibt. Dabei wird ein Byte mehr reserviert, als Zeichen in der Zeile enthalten sind. Der Grund dafür ist, dass das Auslesen von Zeilen, die genau ein Zeichen enthalten, ansonsten nicht funktionieren würde (was wahrscheinlich daran liegt, dass in den in C++ geschriebenen API-Funktionen Strings mit einem 0-Zeichen abgeschlossen sind). Der reservierte Speicher muss mit dem niedrigen und dem hohen Wort der Länge beginnen, wahrscheinlich weil die Länge von der `TextBox`, die diese Nachricht ja empfängt, aus den ersten zwei Bytes des übergebenen Puffers ausgelesen wird. Deshalb kopiert `GetDisplayedTextBoxRows` diese Information über `Marshal.Copy` in die ersten zwei Bytes des Puffers. Die Bytes, die dem hohen und dem niedrigen Wort der Länge entsprechen, wurden zuvor über `BitConverter.GetBytes` ermittelt. Danach wird die Zeile über

SendMessage mit der Nachricht EM_GETLINE eingelesen. SendMessage schreibt die Bytes der Zeile in das Byte-Array, das am letzten Argument übergeben wird. Schließlich kopiert GetDisplayedTextBoxRows den unverwalteten Puffer in ein verwaltetes Byte-Array, konvertiert dieses in einen String, hängt den String an die Ergebnis-Auflistung an und gibt den unverwalteten Speicher wieder frei.

Zum Kompilieren dieser Methode müssen Sie die Namensräume System, System.Collections.Generic, System.Collections.ObjectModel, System.Text, System.Windows.Forms und System.Runtime.InteropServices importieren.

```
/* Deklaration der benötigten API-Funktion */
[DllImport("User32.Dll")]
private static extern int SendMessage(IntPtr hWnd, int msg,
     IntPtr wParam, IntPtr lParam);

/* Gibt die tatsächlich angezeigten Zeilen einer TextBox zurück */
public static ReadOnlyCollection<string> GetDisplayedTextBoxRows(
   TextBox textBox)
{
   // Ergebnis-Auflistung erzeugen
   List<string> result = new List<string>();

   // Konstanten für SendMessage
   const int EM_GETLINECOUNT = 0x00BA;
   const int EM_LINELENGTH = 0x00C1;
   const int EM_LINEINDEX = 0x00BB;
   const int EM_GETLINE = 0x00C4;

   // Anzahl der Zeilen ermitteln
   int lineCount = SendMessage(textBox.Handle, EM_GETLINECOUNT,
      IntPtr.Zero, IntPtr.Zero);

   for (uint lineIndex = 0; lineIndex < lineCount; lineIndex++)
   {
      // Die Position des ersten Zeichens der Zeile ermitteln
      int pos = (int)SendMessage(textBox.Handle, EM_LINEINDEX,
         (IntPtr)lineIndex, IntPtr.Zero);

      // Die Länge der Zeile ab dieser Position ermitteln
      int lineLength = (int)SendMessage(textBox.Handle, EM_LINELENGTH,
         (IntPtr)pos, IntPtr.Zero);

      if (lineLength > 0)
      {
```

Listing 461: Methode zum Auslesen der aktuell angezeigten Zeilen in einer TextBox

```
// Wenn die Zeile Zeichen enthält:
// Platz für das abschließende 0-Zeichen berücksichtigen
lineLength++;

IntPtr apiBuffer = IntPtr.Zero;
try
{
   // Speicher im nicht verwalteten Bereich
   // mit der Länge der Zeile reservieren
   apiBuffer = Marshal.AllocHGlobal(lineLength);

   // Die Länge der Zeile muss laut der Dokumentation in die ersten
   // beiden Bytes des Puffers geschrieben werden, also:
   // über BitConverter.GetBytes die Länge in ein Byte-Array
   // lesen und dieses in die ersten zwei Bytes des Puffers
   // kopieren
   byte[] lengthInfo = BitConverter.GetBytes((short)lineLength);
   Marshal.Copy(lengthInfo, 0, apiBuffer, 2);

   // Die Zeile einlesen
   SendMessage(textBox.Handle, EM_GETLINE,
     (IntPtr)lineIndex, apiBuffer);

   // Den unverwalteten Puffer in ein verwaltetes
   // Byte-Array kopieren
   byte[] clrBuffer = new byte[lineLength];
   Marshal.Copy(apiBuffer, clrBuffer, 0, lineLength);
   string row = String.Empty;
   // Hier wird nicht über Encoding.xyz.GetString kopiert, weil
   // dabei verschiedene Probleme auftraten, wie z. B., dass
   // Umlaute nicht korrekt funktionierten oder dass Pluszeichen
   // zu Problemen führten. Ein Kopieren der einzelnen Zeichen
   // funktioniert aber. Dank an Axel Seibel, der mir die Fehler
   // und die Lösung gemeldet hat.
   for (int i = 0; i < clrBuffer.Length; i++)
   {
      row += Convert.ToChar(clrBuffer[i]);
   }
   if (row.EndsWith("\0"))
   {
      row = row.Remove(row.Length - 1, 1);
   }
   result.Add(row);
}
finally
```

Listing 461: Methode zum Auslesen der aktuell angezeigten Zeilen in einer TextBox (Forts.)

```
            {
               if (apiBuffer != IntPtr.Zero)
               {
                  // Puffer-Speicher freigeben
                  Marshal.FreeHGlobal(apiBuffer);
               }
            }
         }
         else
         {
            // Die Zeile war leer, also eine leere Zeile hinzufügen,
            // allerdings nur dann, wenn die TextBox nicht leer ist
            // (also nur eine virtuelle Zeile enthält)
            if (lineCount != 1)
            {
               result.Add(String.Empty);
            }
         }
      }

      // Ergebnis zurückgeben
      return new ReadOnlyCollection<string>(result);
   }
```

Listing 461: Methode zum Auslesen der aktuell angezeigten Zeilen in einer TextBox (Forts.)

Abbildung 137 zeigt eine Beispielanwendung, in der der automatisch umbrochene Text einer `TextBox` in ein `Label` geschrieben wurde.

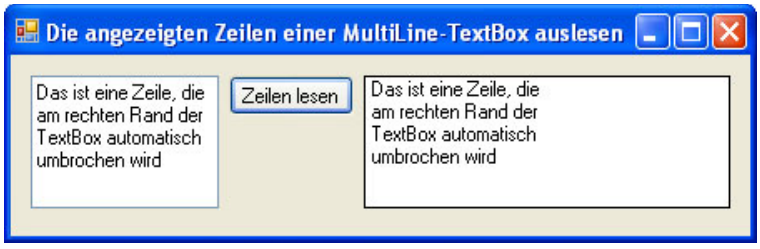


Abbildung 137: In einer Beispielanwendung wurden die Zeilen einer TextBox in ein Label geschrieben

245 ComboBox mit Autovervollständigung

Ein `ComboBox`-Steuerelement, das den zur Anwendereingabe nächsten passenden Listeneintrag sucht und (unter anderem) die Eingabe nach rechts mit einer Selektion versehen entsprechend erweitert (siehe Abbildung 138), ist für Eingaben sehr hilfreich.



Abbildung 138: ComboBox mit Autovervollständigung (Append-Modus)

Ab .NET 2.0 besitzt die `ComboBox` Eigenschaften, über die Sie eine automatische Vervollständigung steuern können. Die `ComboBox` bietet damit einige Möglichkeiten, wie z. B. dass die Liste für die Vervollständigung aus der Liste der `ComboBox`, aus einer separaten Liste oder aus dem Dateisystem gelesen wird. Leider funktioniert das, was der Anwender wohl in 99 Prozent der Fälle benötigt, nämlich das Vorschlagen des nächsten passenden Eintrags aus der ComboBox-Liste, nicht wie erwartet. Weil ich unten im Rezept eine eigene Lösung vorschlage, nenne ich zunächst die Probleme der Autovervollständigung der `ComboBox`:

Hinweis

- Wenn Sie als Quelle der Autovervollständigungs-Liste die `ComboBox`-Liste verwenden, kann es sein, dass bei einer Eingabe nicht der erste passende Eintrag der `ComboBox`-Liste vorgeschlagen wird. Dies ist dann der Fall, wenn die `ComboBox`-Liste eine andere Sortierung aufweist, als eine Sortierung der dargestellten Strings ergeben würde. Die Autovervollständigungs-Liste besteht nämlich aus Strings und wird von der `ComboBox` vor der Anzeige sortiert. Speichert die `ComboBox` z. B. die Versionswerte 1.0.0.5, 1.0.0.100 und 1.0.0.101, schlägt die Autovervollständigung den Wert 1.0.0.100 vor, wenn der Benutzer eine 1 eingibt. Eigentlich sollte aber der erste passende Wert der Liste, nämlich 1.0.0.5 (der ja als Version gesehen kleiner ist als 1.0.0.100), vorgeschlagen werden.
- Ist die Eigenschaft `AutoCompleteMode` auf `Suggest` oder `SuggestAppend` eingestellt und `DropDownStyle` auf `Simple`, scheint die `ComboBox` einen Bug zu beinhalten. Wenn nach einer Eingabe die Liste automatisch aufklappt und Sie über die Cursor-Tasten einen der Einträge auswählen, können Sie Ihre Auswahl nicht mit der Return-Taste übernehmen. Der Eintrag im Textfeld der `ComboBox` wird in diesem Fall einfach gelöscht. Sie können Ihre Auswahl lediglich über die Tab-Taste übernehmen. Ich habe diesen Bug gemeldet. Möglicherweise wird dieses Problem im ersten Service Pack des .NET Framework 3.5 gelöst.

Verwenden der Autovervollständigung der Windows.Forms-ComboBox

Die Eigenschaft `AutoCompleteMode` einer `ComboBox` bestimmt, ob und auf welche Weise die automatische Vervollständigung angewendet wird. Die Einstellung `Suggest` gibt an, dass die `ComboBox` bei Eingaben automatisch eine Liste aufklappt, die Einträge enthält, die mit der Eingabe beginnen (Abbildung 139).



Abbildung 139: ComboBox mit Autovervollständigung im Suggest-Modus

`Append` bewirkt, dass die `ComboBox` die Eingabe auf den nächsten passenden Eintrag erweitert und den hinzugefügten Text automatisch markiert (Abbildung 138). Mit `SuggestAppend` können Sie beide Varianten kombinieren.

Nun müssen Sie über die Eigenschaft `AutoCompleteSource` noch festlegen, aus welchen Daten die Einträge in der Liste entnommen werden. Die wohl gängigste Einstellung, bei der die Einträge aus der `Items`-Auflistung der `ComboBox` entnommen werden, ist `ListItems`. Daneben sind aber auch weitere Einstellungen möglich, die ich in Tabelle 30 beschreibe.

AutoCompleteSource-Einstellung	Bedeutung
`AllSystemSources`	Kombination von `FileSystem` und `AllUrl`
`AllUrl`	Kombination von `HistoryList` und `RecentlyUsedList`
`CustomSource`	Die `ComboBox` verwendet die Strings, die in der Eigenschaft `AutoCompleteCustomSource` verwaltet werden. Diese `StringCollection` muss vom Programmierer gefüllt werden.
`FileSystem`	Die `ComboBox` verwendet das Dateisystem als `AutoComplete`-Liste, ähnlich wie es in der Adress-Box des Windows-Explorers der Fall ist. Geben Sie z. B. »C:\« ein, werden alle Dateien und Ordner, die dem Stammordner des Laufwerks C: untergeordnet sind, in die Liste aufgenommen.
`FileSystemDirectories`	Diese Einstellung funktioniert ähnlich wie `FileSystem`, mit dem Unterschied, dass nur Ordner in die Liste aufgenommen werden.
`HistoryList`	Die `ComboBox` verwendet die History-Liste des Internet Explorers als `AutoComplete`-Liste.
`ListItems`	gibt an, dass die Einträge der `Items`-Auflistung der `ComboBox` für die `AutoComplete`-Liste verwendet werden.
`None`	gibt an, dass keine Quelle für die `AutoComplete`-Liste verwendet wird.
`RecentlyUsedList`	Die `ComboBox` verwendet die Liste der am meisten verwendeten URLs des Internet Explorers als `AutoComplete`-Liste.

Tabelle 30: Mögliche Werte der AutoCompleteSource-Eigenschaft der ComboBox

Hinweis

Für alle Einstellungen außer `None` und `ListItems` macht es wohl keinen Sinn die `ComboBox` im normalen Sinn, also mit einer Liste von Einträgen, die über den Pfeil-Schalter ausgewählt werden können, zu verwenden. Technisch ist dies aber möglich: Sie können der `Items`-Auflistung beliebige Einträge hinzufügen und für die Auto-Vervollständigung zum Beispiel das Dateisystem verwenden. Ich denke aber, dass Benutzer in diesen Fällen eher verwirrt werden, da sie in der Liste andere Einträge finden als diese, die automatisch vervollständigt werden. Für diese speziellen Einstellungen denke ich, dass es keinen Sinn macht, die `Items`-Auflistung zu füllen. Außerdem sollte in diesen Fällen die Eigenschaft `DropDownStyle` auf `Simple` eingestellt werden, damit die `ComboBox` keinen Pfeil-Schalter anzeigt und wie eine normale `TextBox` – allerdings eben mit Auto-Vervollständigung – wirkt.

Eine eigene ComboBox mit (korrekt funktionierender) einfacher Autovervollständigung

Die Klasse `AutoCompleteComboBox`, die ich ursprünglich in der ersten Version des Codebook entwickelt habe, implementiert eine korrekt funktionierende Autovervollständigung, bei der die `ComboBox`-Liste als Quelle verwendet wird. Die Autovervollständigungsliste entspricht damit in der Sortierung immer der Liste der `ComboBox`.

Hinweis

Das einzige Problem, das `AutoCompleteComboBox` hat, ist, dass die von der Klasse `ComboBox` geerbten AutoComplete-Eigenschaften ebenfalls zur Verfügung stehen, da diese leider nicht versteckt werden können.

Die `AutoCompleteComboBox` erfordert den Import der Namensräume `System` und `System.Windows.Forms`.

```
public class AutoCompleteComboBox : ComboBox
{
   /* Privates Feld zur Vermeidung rekursiver Aufrufe
      des TextChanged-Ereignisses */
   private bool dontHandleTextChanged = false;

   /* Verwaltet den Index des aktuellen Treffers */
   private int currentMatchIndex = -1;

   /* Verwaltet den aktuell vom Benutzer eingegebenen Text */
   private string currentInput = null;

   /* OnTextChanged wird überschrieben, um bei einer Änderung des Textes
      den ersten passenden Eintrag zu suchen und den Text zu erweitern */
   protected override void OnTextChanged(EventArgs e)
   {
      if (this.dontHandleTextChanged == false)
```

Listing 462: Eigene ComboBox mit Autovervollständigung

```
      {
         // Die aktuelle Eingabe merken
         this.currentInput = this.Text;

         // Nach einem Eintrag suchen, der am Anfang dem eingegebenen
         // Text entspricht
         this.currentMatchIndex = this.FindString(this.currentInput);

         // Den aktuellen Treffer selektieren
         this.SelectCurrentMatch();
      }

      // Die geerbte Methode aufrufen
      base.OnTextChanged(e);
   }

   /* OnKeyDown wird überschrieben, um die Verarbeitung von OnTextChanged zu
      vermeiden, wenn die Backspace-, die Löschen-, die Cursor-Up- oder die
      Cursor-Down-Taste betätigt wurde. Außerdem behandelt OnKeyDown die
      Betätigung der Cursor-Up- und der Cursor-Down-Taste, um den
      vorherigen bzw. nächsten passenden Eintrag auszuwählen, und die
      Betätigung der Backspace oder Löschen-Taste, um die intern
      verwaltete aktuelle Eingabe zurückzusetzen. */
   protected override void OnKeyDown(KeyEventArgs e)
   {
      // Verarbeitung vermeiden, wenn die Backspace- oder die Löschen-
      // Taste betätigt wurde
      this.dontHandleTextChanged = (e.KeyCode == Keys.Back ||
         e.KeyCode == Keys.Delete || e.KeyCode == Keys.Up ||
         e.KeyCode == Keys.Down);

      int newMatchIndex = -1;
      switch (e.KeyCode)
      {
         case Keys.Up:
            if (this.currentMatchIndex > -1)
            {
               // Ermitteln, ob der Eintrag über dem aktuellen
               // zu der aktuellen Eingabe passt
               if (this.currentMatchIndex > 0)
               {
                  if (this.Items[this.currentMatchIndex - 1].ToString()
                     .StartsWith(this.currentInput,
                     StringComparison.CurrentCultureIgnoreCase))
                  {
```

Listing 462: Eigene ComboBox mit Autovervollständigung (Forts.)

```
                newMatchIndex = this.currentMatchIndex - 1;
             }
          }
       }
       break;

    case Keys.Down:
       if (this.currentMatchIndex > -1)
       {
          // Ermitteln, ob der Eintrag unter dem aktuellen zu der
          // aktuellen Eingabe passt
          if (this.currentMatchIndex < this.Items.Count - 1)
          {
             if (this.Items[this.currentMatchIndex +1]
                .ToString().StartsWith(this.currentInput,
                StringComparison.CurrentCultureIgnoreCase))
             {
                newMatchIndex = this.currentMatchIndex + 1;
             }
          }
       }
       break;

    case Keys.Back:
    case Keys.Delete:
    case Keys.Left:
    case Keys.Right:
       // Bei Backspace, Löschen, Cursor-Links oder Cursor-Rechts
       // die aktuelle Eingabe zurücksetzen
       this.currentInput = null;
       this.currentMatchIndex = -1;
       break;
 }

 if (newMatchIndex > -1)
 {
    // Eintrag gefunden, der zur aktuellen Eingabe passt:
    // Diesen selektieren
    this.currentMatchIndex = newMatchIndex;
    this.SelectCurrentMatch();
 }
```

Listing 462: Eigene ComboBox mit Autovervollständigung (Forts.)

```
      if (newMatchIndex > -1 ||
         ((e.KeyCode == Keys.Up || e.KeyCode == Keys.Down) &&
         this.currentMatchIndex > -1 && (this.SelectionLength > 0 &&
         this.SelectionLength < this.Text.Length)))
      {
         // Wenn ein neuer Treffer ermittelt wurde, oder wenn der Benutzer
         // die Cursor-Up- oder die Cursor-Down-Taste betätigt hat, während
         // aktuell ein Vorschlag angezeigt wird, aber nur Textteile
         // selektiert sind: Das Ereignis als behandelt
         // kennzeichnen, damit die ComboBox die Tasten-Betätigung
         // nicht auswertet.
         e.Handled = true;
      }

      // Die geerbte Methode aufrufen
      base.OnKeyDown(e);
   }

   /* Schreibt den aktuellen Treffer in die TextBox und selektiert den von der
     aktuellen Eingabe differenten Teil */
   private void SelectCurrentMatch()
   {
      if (this.currentMatchIndex >= 0)
      {
         // Eintrag gefunden: Alten Text merken, neuen Eintrag
         // auswählen und den differenten Teil selektieren
         try
         {
            this.dontHandleTextChanged = true;
            this.SelectedIndex = -1;
            this.SelectedIndex = this.currentMatchIndex;
         }
         finally
         {
            this.dontHandleTextChanged = false;
         }
         this.Select(this.currentInput.Length, this.Text.Length);
      }
   }
}
```

Listing 462: Eigene ComboBox mit Autovervollständigung (Forts.)

246 Daten neben den Einträgen einer ListBox oder ComboBox verwalten

Häufig besteht der Bedarf, neben den Einträgen in einer `ListBox` oder `ComboBox` weitere Daten zu verwalten. Das ist z. B. dann der Fall, wenn Sie Datensätze aus einer Datenbanktabelle auslesen und in einer `ListBox` zur Auswahl darstellen. Um den ausgewählten Datensatz eindeutig identifizieren zu können, müssen Sie den Primärschlüssel des Datensatzes neben dem Listeneintrag speichern (wenigstens dann, wenn Sie diese nicht mit anzeigen wollen).

Die Lösung dieses Problems ist sehr einfach: Erstellen Sie für die Listeneinträge eine Klasse oder Struktur, deren `ToString`-Methode den Text zurückgibt, der in der Liste angezeigt werden soll. In den Eigenschaften der Klasse/Struktur können Sie natürlich beliebige Daten verwalten. Beim Einlesen der Liste erzeugen Sie für jeden Eintrag eine Instanz und hängen diese der Liste an. Beim Auslesen der Liste können Sie das Objekt am jeweiligen Eintrag wieder in seinen Typ casten und somit auf die Eigenschaften zugreifen.

Um dies an einem Beispiel zu erläutern, sollen Personendaten in einer `ListBox` ausgegeben werden. Dazu habe ich die Klasse `Person` implementiert:


```
private class Person
{
   public int Id;
   public string FirstName;
   public string LastName;

   /* Konstruktor */
   public Person(int id, string firstName, string lastName)
   {
      this.Id = id;
      this.FirstName = firstName;
      this.LastName = lastName;
   }

   /* Die ToString-Methode wird überschrieben */
   public override string ToString()
   {
      return this.FirstName + " " + this.LastName;
   }
}
```

Listing 463: Klasse für die Speicherung von Personendaten zur Ausgabe in einer ListBox oder ComboBox

Im `Load`-Ereignis eines Formulars werden Personendaten aus einer Datenbank eingelesen und in eine `ListBox` geschrieben (was hier nur simuliert wird):

```
private void StartForm_Load(object sender, System.EventArgs e)
{
   // Einige Personen an die ListBox anhängen
   this.personList.Items.Add(new Person(1001, "Zaphod", "Beeblebrox"));
   this.personList.Items.Add(new Person(1002, "Tricia", "McMillan"));
   this.personList.Items.Add(new Person(1003, "Arthur", "Dent"));
   this.personList.Items.Add(new Person(1004, "Ford", "Prefect"));
   this.personList.Items.Add(new Person(1005, "Marvin", ""));
}
```

Listing 464: Füllen der Liste mit neuen Person-Objekten

Bei der Auswahl eines Eintrags liest das Beispiel das aktuelle Objekt aus der Liste aus und stellt dessen Daten in einem `Label` dar:

```
private void personList_SelectedIndexChanged(object sender,
   System.EventArgs e)
{
   // Auslesen des selektierten Objekts
   Person p = (Person)personList.SelectedItem;
   this.infoLabel.Text = "Id: " + p.Id + "\r\n" +
      "Vorname: " + p.FirstName + "\r\n" +
      "Nachname: " + p.LastName;
}
```

Listing 465: Auslesen des gewählten Objekts bei der Änderung des selektierten Eintrags

Das Ergebnis zeigt Abbildung 140.

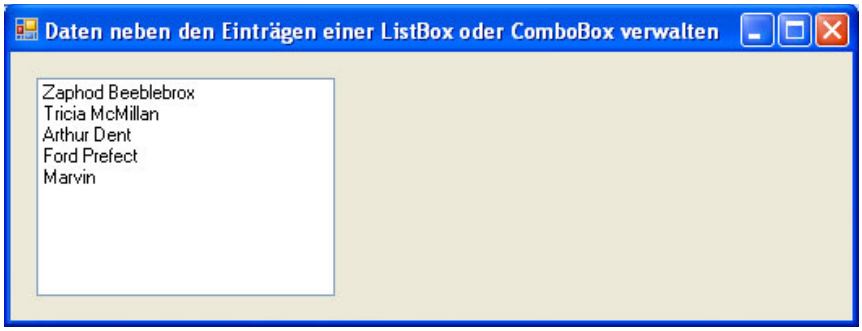


Abbildung 140: Das Beispielprogramm in Aktion

247 ListBox mit ToolTip für längere Einträge

Werden in einer `ListBox` Einträge verwaltet, die länger sind als die Breite der `ListBox`, werden diese leider nur abgeschnitten dargestellt. Das ist immer sehr ärgerlich, denn dabei gehen Informationen verloren. Für den Benutzer wäre es hilfreich, wenn zu lange Einträge in einem ToolTip angezeigt werden würden, sobald die Maus auf diesen Einträgen verweilt.

Um dieses Problem zu lösen habe ich die ListBox in einer eigenen Klasse ToolTipListBox erweitert. Listing 466 zeigt diese Klasse. In der überschriebenen OnMouseMove-Methode ermittelt die ToolTipListBox zunächst den Eintrag, der unter dem Cursor liegt. Über die MeasureString-Methode des Graphics-Objekts, das der ListBox zugeordnet ist, berechnet OnMouseMove dann die Breite des Eintrags. Dabei wird das StringFormat-Objekt übergeben, das die StringFormat.GenericTypographic-Eigenschaft zurückgibt, um die Breite typografisch korrekt zu berechnen. Die berechnete Breite wird mit der Breite des Client-Rechtecks der ListBox verglichen. Ist der Eintrag zu breit, wird der ToolTip, der in dem privaten Feld toolTip verwaltet wird, mit dem Eintrag gesetzt. Im anderen Fall wird der ToolTip zurückgesetzt. Um den ToolTip auch dann zurückzusetzen, wenn die Maus die ListBox verlässt, überschreibt ToolTipListBox schließlich noch die OnMouseLeave-Methode.

Zum Kompilieren dieser Klasse müssen Sie die Namensräume System, System.Windows.Forms und System.Drawing importieren.

```
public class ToolTipListBox : ListBox
{
   /* Verwaltet den ToolTip */
   private ToolTip toolTip = new ToolTip();

   /* Setzt den ToolTip mit dem Eintrag, der gerade unter der Maus liegt,
      wenn der Eintrag nicht komplett angezeigt wird */
   protected override void OnMouseMove(MouseEventArgs e)
   {
      base.OnMouseMove(e);

      // Ermitteln des Eintrags unter der Maus
      Point p = base.PointToClient(Cursor.Position);
      int index = base.IndexFromPoint(p);
      if (index > -1)
      {
         string item = base.Items[index].ToString();

         // Überprüfen, ob der Eintrag in der Breite in die Liste passt
         using (Graphics g = base.CreateGraphics())
         {
            int width = (int)g.MeasureString(item, base.Font,
               0, StringFormat.GenericTypographic).Width;
            if (width > base.ClientRectangle.Width)
            {
               // ToolTip setzen
               this.toolTip.SetToolTip(this, item);
            }
            else
            {
```

Listing 466: Klasse für eine ListBox, die bei zu langen Einträgen automatisch einen ToolTip anzeigt

```
                // ToolTip zurücksetzen
                this.toolTip.SetToolTip(this, null);
            }
        }
    }
    else
    {
        // ToolTip zurücksetzen
        this.toolTip.SetToolTip(this, null);
    }
}

/* Setzt den ToolTip wieder zurück */
protected override void OnMouseLeave(EventArgs e)
{
    base.OnMouseLeave(e);

    // ToolTip zurücksetzen
    this.toolTip.SetToolTip(this, null);
}
}
```

Listing 466: Klasse für eine ListBox, die bei zu langen Einträgen automatisch einen ToolTip anzeigt (Forts.)

Abbildung 141 zeigt eine Beispiel-Anwendung.

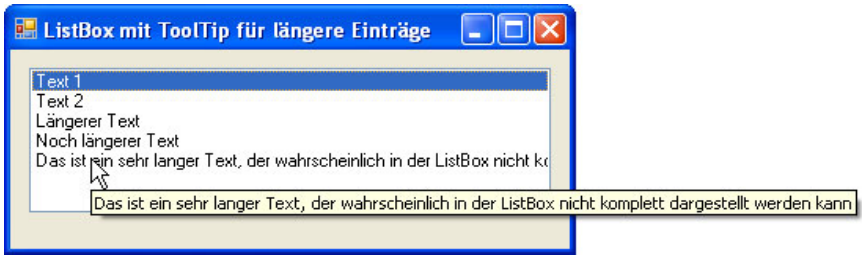


Abbildung 141: Beispielanwendung mit einer ListBox, die längere Einträge in einem ToolTip anzeigt

248 ListView sortieren

`ListView`-Steuerelemente lassen sich in vielen Anwendungen in der Detailansicht durch einen Klick auf eine Spalte sortieren. Standardmäßig sortiert ein `ListView`-Steuerelement die Einträge aber nur nach der ersten Spalte, wenn Sie in der Eigenschaft `Sorting` den Wert `SortOrder.Ascending` oder `SortOrder.Descending` angeben.

Sie können aber stattdessen auch `Sorting` auf `SortOrder.None` einstellen und zum Sortieren die `Sort`-Methode aufrufen. `Sort` sortiert die Liste nach dem Objekt, das der Eigenschaft `ListView-`

ItemSorter zugewiesen ist und das die Schnittstelle IComparer implementiert. Und das ist der Trick.

Abbildung 142 zeigt ein Beispielformular mit einem ListView-Steuerelement mit Namen *bookList*. Die einzelnen Spalten sollen bei einem Klick auf die Überschrift automatisch sortiert werden. Das Beispiel ist nach der ISBN-Nummer sortiert.

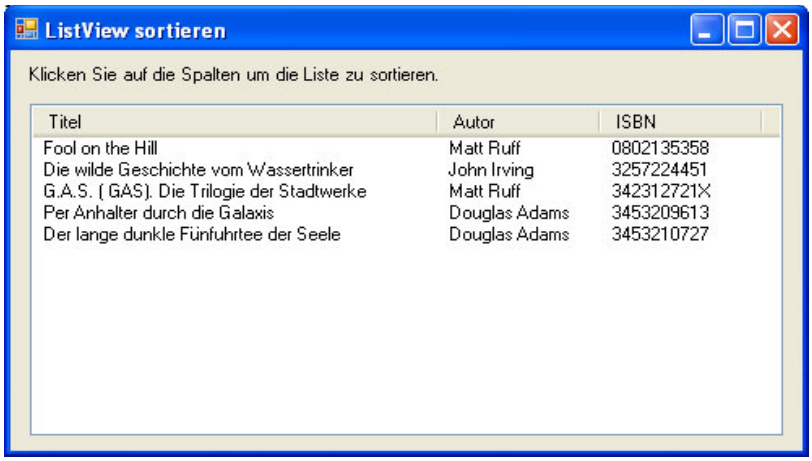


Abbildung 142: Beispielformular für das Sortieren eines ListView-Steuerelements

Zur Lösung des Sortierproblems können Sie nun eine Klasse implementieren, die die Compare-Methode der Schnittstelle IComparer implementiert:

```
/* Klasse, die IComparer implementiert und die das Vergleichen von
   ListView-Elementen übernimmt */
public class ListViewItemComparer : IComparer
{
   /* Verwaltet den Index der aktuellen Spalte */
   public int CurrentColumn = 0;

   /* Vergleicht zwei ListViewItem-Objekte */
   public int Compare(object x, object y)
   {
      ListViewItem firstItem = (ListViewItem)x;
      ListViewItem secondItem = (ListViewItem)y;

      return string.Compare(
         firstItem.SubItems[CurrentColumn].Text,
         secondItem.SubItems[CurrentColumn].Text);
   }
}
```

Listing 467: Klasse zum Sortieren eines ListView-Steuerelements

Das Feld `CurrentColumn` nimmt später den Index der aktuell angeklickten Spalte auf. Über diesen Index vergleicht die `Compare`-Methode die Spaltenwerte der übergebenen `ListViewItem`-Objekte. `Compare` muss einen Wert kleiner Null zurückgeben, wenn das x kleiner y ist, einen Wert größer Null, wenn x größer y ist, und Null, wenn beide Objekte gleich sind. Da die `Compare`-Methode der `String`-Klasse einen passenden Vergleich liefert, ruft `Compare` einfach diese Methode auf.

Im Formular benötigen Sie dann ein privates Feld, das eine Instanz der `ListViewItemComparer`-Klasse aufnimmt:

```
private ListViewItemComparer listViewItemComparer =
   new ListViewItemComparer();
```

Im `Load`-Ereignis (oder wo auch immer Sie das `ListView`-Steuerelement initialisieren) weisen Sie dem Feld `listViewItemComparer` die Referenz auf die `ListViewItemComparer`-Instanz zu:

```
private void StartForm_Load(object sender, System.EventArgs e)
{
   // Füllen des ListView-Steuerelements
   this.bookList.Items.Add(new ListViewItem(new string[] {
      "Fool on the Hill", "Matt Ruff", "0802135358"}));
   this.bookList.Items.Add(new ListViewItem(new string[] {
      "G.A.S. ( GAS). Die Trilogie der Stadtwerke", "Matt Ruff",
      "342312721X"}));
   this.bookList.Items.Add(new ListViewItem(new string[] {
      "Per Anhalter durch die Galaxis", "Douglas Adams",
      "3453209613"}));
   this.bookList.Items.Add(new ListViewItem(new string[] {
      "Der lange dunkle Fünfuhrtee der Seele", "Douglas Adams",
      "3453210727"}));
   this.bookList.Items.Add(new ListViewItem(new string[] {
      "Die wilde Geschichte vom Wassertrinker", "John Irving",
      "3257224451"}));

   // Comparer zuweisen
   this.listViewItemComparer.CurrentColumn = 2; // Sortieren nach ISBN
   this.bookList.ListViewItemSorter = this.listViewItemComparer;
}
```

Listing 468: Initialisieren des ListView-Steuerelements

Nun müssen Sie nur noch den Klick auf eine Spalte auswerten. Dazu können Sie das `ColumnClick`-Ereignis verwenden. Hier stellen Sie die aktuelle Spalte für Ihr `ListViewItemComparer`-Objekt ein und rufen die `Sort`-Methode des `ListView`-Steuerelements auf:

```
private void bookList_ColumnClick(object sender, ColumnClickEventArgs e)
{
   // Umsortieren der Liste beim Klick auf eine Spalte
   this.listViewItemComparer.CurrentColumn = e.Column;
   this.bookList.Sort();
}
```

Listing 469: Umsortieren der Liste beim Klick auf eine Spalte

Fertig ist das Sortieren.

249 Knoten einer Ebene in einem TreeView-Steuerelement vertauschen

Für die Implementierung spezieller Features wie z. B. der Möglichkeit für den Anwender, die Knoten eines `TreeView`-Steuerelements definiert zu verschieben, ist es notwendig, zwei Knoten einer Ebene gegeneinander zu tauschen. Dazu können Sie aber nicht einfach nur die Referenzen der `TreeNode`-Objekte tauschen, wie es das folgende Beispiel zeigt, das versucht, die ersten beiden Unterknoten des ersten Hauptknotens des `TreeView` ***bookTree*** zu vertauschen:


```
TreeNode node = bookTree.Nodes[0].Nodes[0];
bookTree.Nodes[0].Nodes[2] = bookTree.Nodes[0].Nodes[1];
bookTree.Nodes[0].Nodes[1] = node;
```

Listing 470: Fehlerhaftes Vertauschen zweier TreeView-Knoten

Bei diesem Versuch gerät das `TreeView`-Steuerelement durcheinander und zeigt Einträge auf einmal doppelt an, obwohl die Anzahl der `TreeNode`-Objekte dieselbe ist wie vorher. Abbildung 144 zeigt eine Beispielanwendung mit zuvor eindeutigen Einträgen (Abbildung 143) nach dem dreimaligen Aufruf der oben beschriebenen Tausch-Variante.

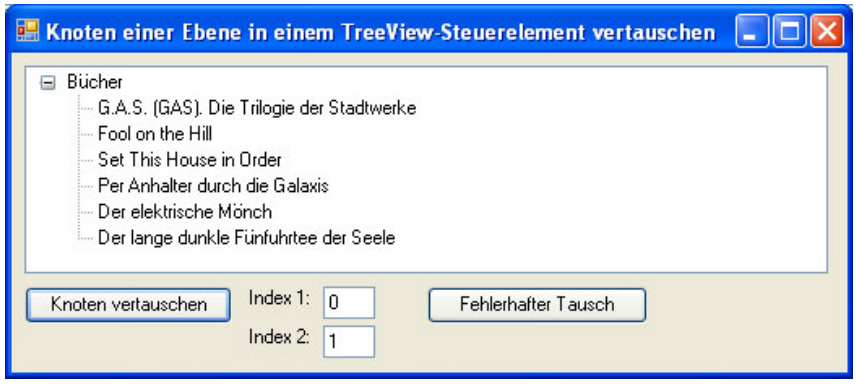


Abbildung 143: Das Beispielprogramm vor dem Vertauschen

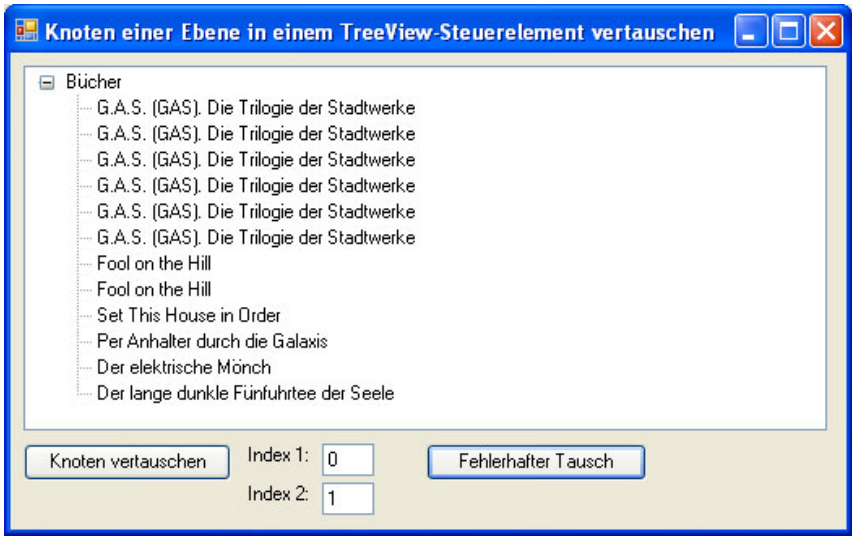


Abbildung 144: Fehlerhafte Darstellung eines TreeView-Steuerelements nach drei inkorrekten Tauschvorgängen

Das Problem liegt darin begründet, dass jeder Knoten Referenzen zu seinem Vorgänger und seinem Nachfolger verwaltet. Und diese müssten Sie ebenfalls umbiegen, was in der Praxis nur sehr schwierig zu programmieren ist.

Die Lösung des Problems verwendet die Methoden `RemoveAt` und `Insert` der `TreeNodeCollection`-Instanz, deren Unterknoten getauscht werden sollen.

Um sicherzustellen, dass der Tausch bei gültigen Indizes immer funktioniert, vergleicht die Methode `SwapTreeViewNodes` in Listing 471 zunächst, ob die beiden Indizes gleich sind (in dem Fall darf und muss nichts gemacht werden). Zur Sicherheit werden die Indizes anschließend noch so umgebogen, dass in `i1` der kleinere und in `i2` der größere gespeichert ist. Dann können die Knoten gefahrlos entfernt und wieder neu eingefügt werden.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System` und `System.Windows.Forms` importieren.

```
public static void SwapTreeViewNodes(TreeNodeCollection nodes, int index1,
   int index2)
{
   if (index1 != index2)
   {
      int i1 = Math.Min(index1, index2);
      int i2 = Math.Max(index1, index2);
      TreeNode node1 = nodes[i1];
      TreeNode node2 = nodes[i2];
      nodes.RemoveAt(i1);
```

Listing 471: Methode zum Vertauschen zweier TreeView-Knoten

```
      nodes.RemoveAt(i2-1);
      nodes.Insert(i1, node2);
      nodes.Insert(i2, node1);
   }
}
```

Listing 471: Methode zum Vertauschen zweier TreeView-Knoten (Forts.)

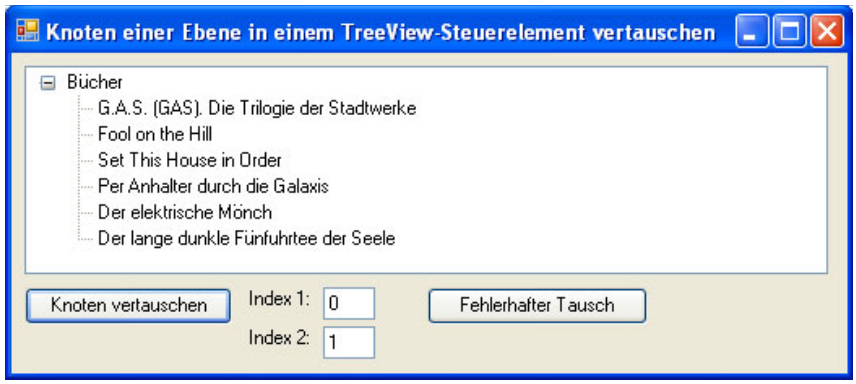


Abbildung 145: Die Beispielanwendung nach dem Vertauschen der oberen beiden Einträge

250 Einzelne Knoten eines TreeView-Steuerelements sortieren

Ein `TreeView`-Steuerelement erlaubt über die `Sorted`-Eigenschaft das generelle Sortieren nach den Knoten. Dabei werden allerdings immer alle Ebenen sortiert. Wenn Sie in einem `TreeView`-Steuerelement nur bestimmte Unterknoten bzw. nur bestimmte Ebenen sortieren wollen, müssen Sie etwas anders vorgehen.

In den C#-Newsgroups fand ich dazu (als ich diese Zeilen ursprünglich schrieb und bei der Überarbeitung für das .NET Framework 2.0 ...) recht wenig Anregungen und schon gar keine fertigen Lösungen. Die einzige dort angedeutete Lösung ist der Aufruf der API-Funktion `SendMessage` mit der Nachricht `TVM_SORTCHILDRENCB`, die genau dazu vorgesehen ist. Leider ist diese wahrscheinlich sehr schnelle Lösung nicht einfach umzusetzen, weswegen ich im Buch darauf verzichte. Nur damit Sie wissen, worum es dabei geht:

Bei dieser Lösung müssen Sie eine Callback-Methode zur Verfügung stellen, die drei `IntPtr`-Argumente übergeben bekommt. Die Adresse dieser Methode übergeben Sie über eine Struktur vom Typ `TVSORTCB` und einen Delegaten neben dem Handle des zu sortierenden Knotens und der Nachricht an `SendMessage`. Der Knoten ruft dann zur Sortierung für jeweils zwei Unterknoten die Callback-Methode auf. In dieser vergleichen Sie die beiden ersten übergebenen `IntPtr`-Werte (der dritte ist für benutzerdefinierte Zwecke vorgesehen). Ist der linke größer als der rechte, muss die Methode einen Wert kleiner Null zurückgeben, ist der rechte größer als der linke, muss ein Wert größer Null zurückgegeben werden, sind beide gleich groß, muss die Methode Null zurückgeben. So weit, so gut. Leider werden in den Argumenten aber keine Referenzen auf die Knoten übergeben (was prinzipiell gar nicht möglich ist, dass API-Funktionen keine Referenzen kennen),

sondern benutzerdefinierte Daten. Und das ist das Problem, denn diese müssen Sie beim oder nach dem Füllen des `TreeView` in einer `TVITEM`-Struktur über eine `TVM_GETITEM`-Nachricht einlesen und über eine `TVM_SETITEM`-Nachricht in den Knoten zurückschreiben. Im Feld `lParam` der `TVITEM`-Struktur können Sie vor dem Schreiben einen `int`-Wert ablegen, der der Callback-Methode beim Sortieren übergeben wird. Dummerweise können Sie hier nicht einfach den Index des Knotens ablegen (was den Zugriff erleichtern würde), denn dieser wird beim Umsortieren ja verändert. Sie müssen sich einen Algorithmus überlegen, der aus den dargestellten Daten des Knotens einen sortierbaren `int`-Wert generiert. Und das ist meiner Ansicht nach fast unmöglich.

Deshalb habe ich eine einfachere Lösung entwickelt. Die Methode `SortTreeViewNodes` in Listing 472 sortiert eine `TreeNodeCollection` (die Sie über die `Node`-Eigenschaft eines Knotens erreichen) über den guten alten Bubblesort. Das einzige Problem dabei ist, dass zum Vertauschen von zwei `TreeNode`-Objekten nicht einfach nur die Referenzen vertauscht werden können. Wenn Sie das machen, bringen Sie das `TreeView`-Steuerelement durcheinander (siehe Rezept 249). Um dies zu vermeiden ruft `SortTreeViewNodes` zum Tauschen einfach die entsprechenden Methoden der `TreeNode`-Objekte auf.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System` und `System.Windows.Forms` importieren.


```
public static void SortTreeViewNodes(TreeNodeCollection nodes)
{
   // Sortieren der Unterknoten über einen einfachen BubbleSort
   for (int i = 0; i < nodes.Count; i++)
   {
      for (int j = nodes.Count - 1; j > i; j--)
      {
         if (String.Compare(nodes[i].Text, nodes[j].Text, true) > 0)
         {
            // Tauschen
            TreeNode node1 = nodes[i];
            TreeNode node2 = nodes[j];
            nodes.RemoveAt(i);
            nodes.RemoveAt(j-1);
            nodes.Insert(i, node2);
            nodes.Insert(j, node1);
         }
      }
   }
}
```

Listing 472: Methode zum Sortieren einer TreeNodeCollection

Abbildung 146 zeigt eine Anwendung, die die zweite Ebene eines `TreeView`-Steuerelements über `SortTreeViewNodes` sortiert hat. Die Unterknoten des dritten Hauptknotens wurden in dieser Anwendung über einen zufällig ermittelten String erzeugt.



Abbildung 146: TreeView mit einer Sortierung der zweiten Ebene

Die Geschwindigkeit dieser Methode ist bei nicht allzu vielen Unterknoten einigermaßen gut. Bei 100 zufällig ermittelten (und damit sehr unsortierten) Unterknoten benötigte der Vorgang auf einem Pentium-4-Rechner mit 2,4 GHz etwa 0,2 Sekunden, wenn der übergeordnete Knoten zugeklappt war. War dieser jedoch aufgeklappt, benötigte die Methode schon 1,2 Sekunden. Bei 1.000 Unterknoten war die Performance allerdings nicht mehr allzu gut. Hier benötigte `SortTreeViewNodes` stolze 34 Sekunden. Ein Vergleich mit einem ebenfalls implementierten QuickSort brachte übrigens sogar ein schlechteres Ergebnis (66 Sekunden für 1.000 Unterknoten bei zugeklapptem Hauptknoten). Sie finden diese Sortierung im Beispiel zu diesem Rezept. Die Messwerte habe ich übrigens über die Klasse `StopWatch` aus dem Rezept 74 ermittelt.

251 Die Werte einzelner Zellen in einem DataGridView setzen und lesen

Dieses Rezept ist eines der kürzesten in diesem Buch, aber ich denke, es besitzt trotzdem (wenigstens für mich ☺) Berechtigung: Wenn Sie auf einzelne Zellen eines `DataGridView`-Steuerelements zugreifen wollen, finden Sie dazu keine Eigenschaft (ich habe zum Beispiel erfolglos nach einer Eigenschaft *Cells* gesucht. Der Zugriff auf die Zellen erfolgt über den Indexer des Steuerelements. Diesem übergeben Sie den Index der Spalte und den Index der Zeile. Den Index einer bestimmten Spalte können Sie über die `Columns`-Auflistung des `DataGridView`-Steuerelements herausfinden, indem Sie deren Indexer den Namen der Spalte übergeben und die `Index`-Eigenschaft abfragen. Der Indexer des `DataGridView`-Steuerelements gibt eine Referenz auf ein `DataGridViewCell`-Objekt zurück. Über dessen Eigenschaft `Value` können Sie den Wert der Zelle lesen und setzen. Da die Zelle auch leer sein kann, sollten Sie beim Lesen auf `null` und sicherheitshalber auch auf `DBNull.Value` überprüfen:

```
int rowIndex = 0;
string columnName = "unitsInStockColumn";

// Den Index der Spalte ermitteln
int columnIndex = this.productsGridView.Columns[columnName].Index;

// Die Zelle referenzieren, die in der ermittelten
// Spalte und der aktuellen Zeile liegt
DataGridViewCell cell = this.productsGridView[columnIndex, rowIndex];

if (cell.Value != DBNull.Value && cell.Value != null)
{
   // Den in dieser Zelle gespeicherten Wert auslesen ...
   int unitsInStock = Convert.ToInt32(cell.Value);

   // und weiter verarbeiten ...
}
```

Listing 473: Auslesen der Werte einzelner Zellen eines DataGridView-Steuerelements

252 Ein DataGridView anpassen

Das neue `DataGridView`-Steuerelement ermöglicht eine sehr flexible Einstellung der angezeigten Spalten, was sogar über den Visual-Studio-Designer möglich ist, sofern Sie das Steuerelement über eine `BindingSource`-Komponente an Daten binden, die wiederum an eine Projekt-Datenquelle gebunden ist. Die Einstellung der Spalten erreichen Sie über den Smart-Tag des Steuerelements. Abbildung 147 zeigt die Einstellung einer Spalte eines `DataGridView`-Steuerelements, die eine `ComboBox` enthält. Das `DataGridView`-Steuerelement in diesem Beispiel ist an die *Products*-Tabelle der *Northwind*-Datenbank gebunden. Die `ComboBox` zeigt statt der in der Tabelle gespeicherten *CategoryId* den Namen der Kategorie an.

Neben `ComboBox`-Spalten sind noch Spalten möglich, die einen Schalter, eine `CheckBox`, einen Hyperlink, ein Bild oder eine `TextBox` anzeigen. Diese will ich hier aber nicht näher beschreiben, probieren Sie es einfach aus.

Für den Fall, dass Sie das Steuerelement nicht bereits in der Entwurfszeit an Daten binden können, z. B. weil Sie es später an ein `DataTable`-, `DataView`- oder `DataSet`-Objekt binden, können Sie lediglich ungebundene Spalten im Designer hinzufügen. Dabei sollten Sie beachten, da Sie beim Bearbeiten der Spalten mehr Einstellungsmöglichkeiten besitzen als beim Hinzufügen einer neuen Spalte. Im Prinzip können Sie hier alles für die Datenbindung Wichtige definieren. Lediglich wenn Sie die `ComboBox` einer `ComboBox`-Spalte ebenfalls in ein Programm selbst füllen wollen, müssen Sie (natürlich) selbst programmieren.

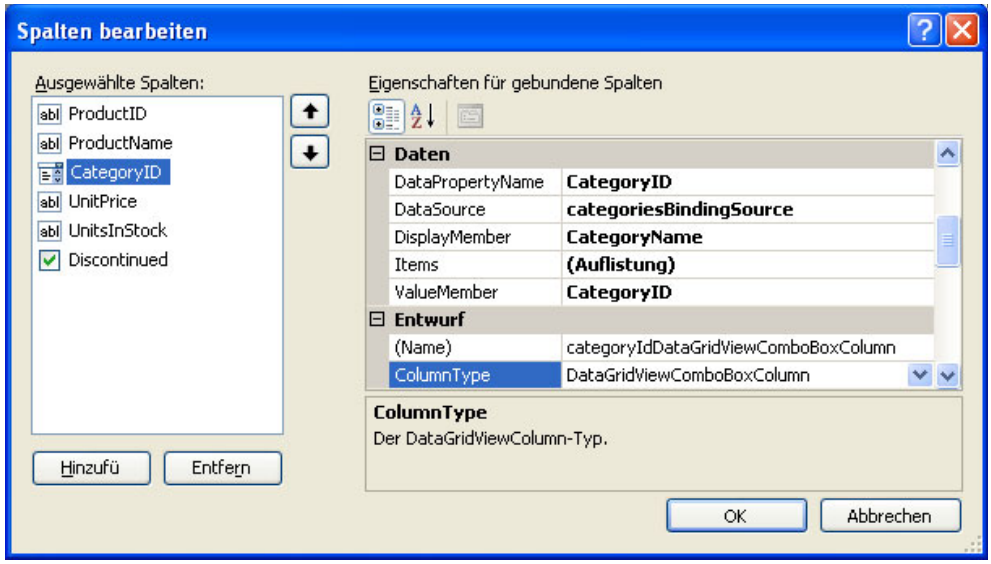


Abbildung 147: Einstellung einer Spalte eines DataGridView-Steuerelements, die die Kategorie eines Produkts der Northwind-Datenbank in einer ComboBox anzeigt

Dazu können Sie die entsprechende Spalte über die `Columns`-Auflistung referenzieren, in den eigentlichen Typ der Spalte casten und die Einstellungen bearbeiten. Im folgenden Beispiel initialisiere ich ein `DataGridView`-Steuerelement, das die Daten von Personen anzeigt. Die Spalten habe ich im Visual-Studio-Designer erzeugt. Bei der dritten Spalte handelt es sich um eine `ComboBox`-Spalte, die den Typ der Person (Erdling oder Alien) verwaltet. Das Beispiel beinhaltet ebenfalls das Füllen der benötigten `DataTable`-Objekte:

```
// Daten für das Personen-DataGridView erzeugen
DataTable personTable = new DataTable();
personTable.Columns.Add("FirstName", typeof(string));
personTable.Columns.Add("LastName", typeof(string));
personTable.Columns.Add("Type", typeof(int));
personTable.Rows.Add(new object[] { "Zaphod", "Beeblebrox", 2 });
personTable.Rows.Add(new object[] { "Ford", "Prefect", 2 });
personTable.Rows.Add(new object[] { "Tricia", "McMillan", 1 });

DataTable personTypeTable = new DataTable();
personTypeTable.Columns.Add("Id", typeof(int));
personTypeTable.Columns.Add("Name", typeof(string));
personTypeTable.Rows.Add(new object[] { 1, "Erdling" });
personTypeTable.Rows.Add(new object[] { 2, "Alien" });
```

Listing 474: Füllen einer ComboBox-Spalte im Programm

```
// Die ComboBox-Spalte des Personen-DataGridView
// initialisieren
DataGridViewComboBoxColumn comboBoxColumn =
   (DataGridViewComboBoxColumn)this.personGridView.Columns[2];
comboBoxColumn.DisplayMember = "Name";
comboBoxColumn.ValueMember = "Id";
comboBoxColumn.DataSource = personTypeTable;
```

Listing 474: Füllen einer ComboBox-Spalte im Programm (Forts.)

Das Ergebnis zeigt Abbildung 148 (inklusive eines `DataGridView`-Steuerelements, das die wichtigsten Daten der Artikel der Northwind-Datenbank anzeigt).

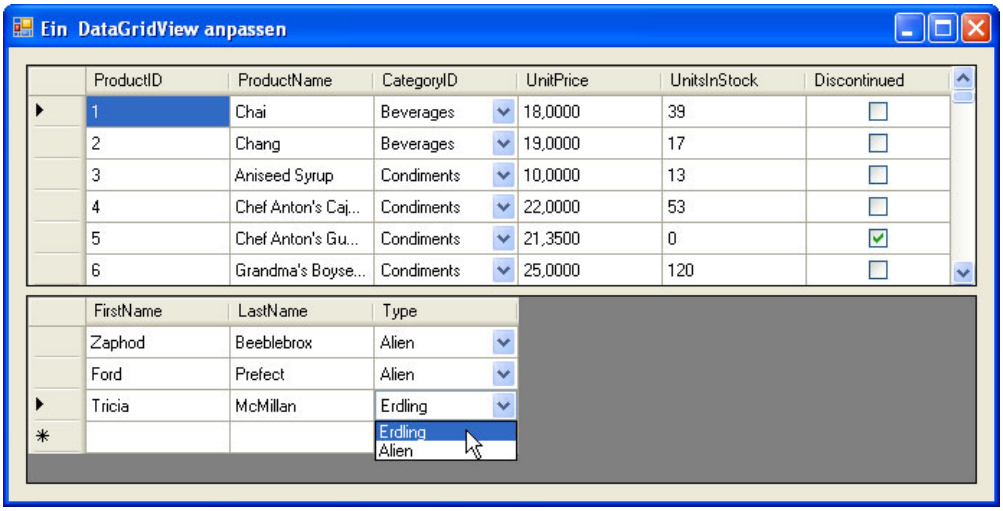


Abbildung 148: Beispiel-Programm mit zwei DataGridView-Instanzen

Bedingungsabhängiges Anpassen der Zellenfarbe

Die Farbe der einzelnen Zeilen eines `DataGridView` können Sie, wie Sie ja wahrscheinlich wissen, über die Eigenschaften `DefaultCellStyle.BackColor` und `AlternatingRowsDefaultTextStyle.BackColor` beeinflussen. Setzen Sie `DefaultCellStyle.BackColor` z. B. auf `Color.White` und `AlternatingRowsDefaultTextStyle.BackColor` auf `Color.Silver`, erscheint jede zweite Zeile hellgrau und jede andere weiß.

Wenn Sie aber Zeilen oder einzelne Zellen anders darstellen wollen, wenn diese einer bestimmten Bedingung entsprechen, müssen Sie etwas mehr programmieren. Das `DataGridView`-Steuerelement stellt Ihnen dazu aber eine Vielzahl an Ereignissen zur Verfügung, die im Prinzip alle Anforderungen abdecken.

Im `CellFormatting`-Ereignis können Sie einzelne Zellen, z. B. abhängig von ihrem Wert, umformatieren. Dazu greifen Sie, wie ich es in Rezept 251 beschrieben habe, über den Indexer des Steuerelements auf die Zelle zu. Den Index der aktuell formatierten Zelle erhalten Sie über die Eigenschaft `Index` des Ereignisargument-Objekts. Den Stil der Zelle können Sie über die

Eigenschaft `CellStyle` beeinflussen. Wollen Sie den Stil der gesamten Zeile beeinflussen, erreichen Sie dies über die Eigenschaft `DefaultCellStyle` der Zeile, die Sie wiederum über die `Rows`-Eigenschaft des `DataGridView`-Steuerelements ansprechen können.

Das folgende Beispiel formatiert die aktuelle Zeile mit einer orangen Hintergrundfarbe, wenn das Feld *Discontinued* `true` speichert. Außerdem wird die Zelle, die das Feld *UnitsInStock* anzeigt, mit einer roten Hintergrundfarbe formatiert, wenn der gespeicherte Wert kleiner/gleich Null ist oder die Zelle nichts speichert.

```
private void productsGridView_CellFormatting(object sender,
   DataGridViewCellFormattingEventArgs e)
{
   // Die gesamte Zeile mit einer orangen Hintergrundfarbe
   // formatieren, wenn die Zelle, die das Feld 'Discontinued'
   //  verwaltet, den Wert true speichert
   if (e.ColumnIndex == this.productsGridView.Columns[
      "discontinuedDataGridViewCheckBoxColumn"].Index)
   {
      // Den Wert des Feldes 'Discontinued' auslesen
      DataGridViewCell cell =
         ((DataGridView)sender)[e.ColumnIndex, e.RowIndex];
      if (cell.Value != null && cell.Value != DBNull.Value)
      {
         bool discontinued = Convert.ToBoolean(cell.Value);
         if (discontinued)
         {
            // Die Hintergrundfarbe der Zeile auf Orange setzen
            ((DataGridView)sender).Rows[e.RowIndex]
               .DefaultCellStyle.BackColor = Color.Orange;
         }
      }
   }

   // Die Zelle, die das Feld 'UnitsInStock' verwaltet,
   // mit einer roten Hintergrundfarbe formatieren, wenn
   // die Zelle nichts oder einen Wert kleiner/gleich Null
   // speichert
   if (e.ColumnIndex == this.productsGridView.Columns[
      "unitsInStockDataGridViewTextBoxColumn"].Index)
   {
      // Den Wert des Feldes 'UnitsInStock' auslesen
      DataGridViewCell cell =
         ((DataGridView)sender)[e.ColumnIndex, e.RowIndex];
      if (cell.Value != null && cell.Value != DBNull.Value)
      {
         int unitsInStock = Convert.ToInt32(cell.Value);
```

Listing 475: Umformatieren von Zeilen und Zellen abhängig vom Inhalt einer Zelle

```
            if (unitsInStock <= 0)
            {
               // Die Hintergrundfarbe der Zelle auf Rot setzen
               e.CellStyle.BackColor = Color.Red;
            }
         }
      }
   }
```

Listing 475: Umformatieren von Zeilen und Zellen abhängig vom Inhalt einer Zelle (Forts.)

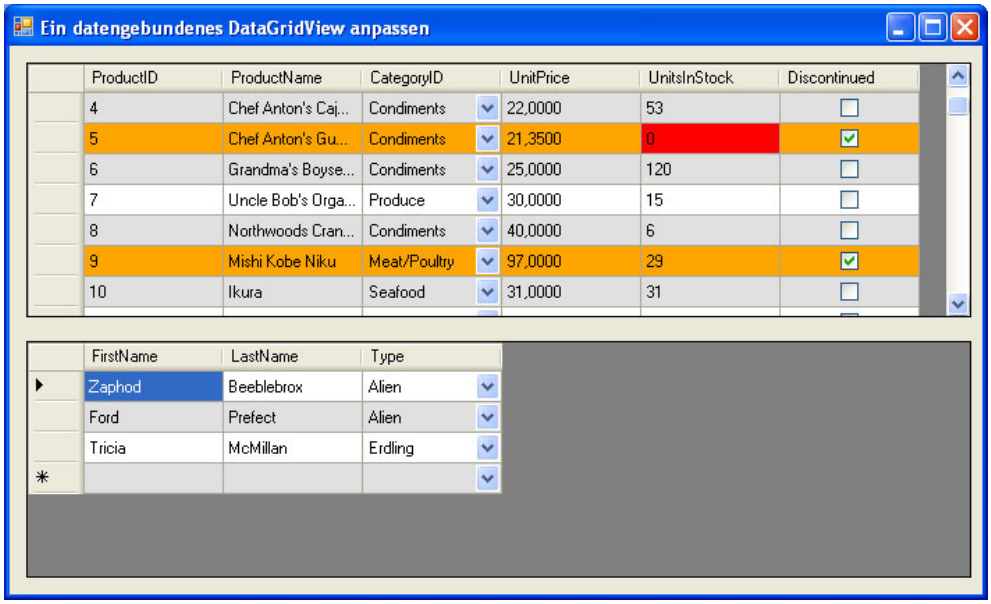


Abbildung 149: Das Beispielprogramm in Aktion

Hinweis

Wenn Sie den Zelleninhalt komplett selbst zeichnen wollen, können Sie dazu das `CellPainting`-Ereignis verwenden. Zeilen können Sie in `RowPrePaint` und `RowPostPaint` benutzerdefiniert zeichnen. Diese Möglichkeiten beschreibe ich hier allerdings nicht.

253 Ein DataGridView mit eigenen Spalten am Beispiel einer DateTimePicker-Spalte

Das `DataGridView`-Steuerelement bietet bereits standardmäßig flexible Möglichkeiten, Daten tabellenförmig darzustellen und dem Anwender zum Bearbeiten zur Verfügung zu stellen. In einigen Fällen wollen Sie aber vielleicht spezielle Spalten integrieren, für die der Designer keinen Spaltentyp anbietet. Ein Beispiel dafür ist eine Spalte, die für ein Datumsfeld ein `DateTimePicker`-Steuerelement anzeigt (Abbildung 150).

Zur Lösung dieses Problems benötigen Sie drei Klassen:

- Eine spezielle Steuerelement-Klasse implementiert das Steuerelement, das im Editiermodus der Zelle angezeigt werden soll. Dieses Steuerelement muss die Schnittstelle `IDataGridViewEditingControl` (aus dem Namensraum `System.Windows.Forms`) implementieren.
- Die zweite Klasse implementiert die Zelle und muss zumindest von `DataGridViewCell` abgeleitet werden. In den meisten Fällen ist es jedoch sinnvoll, diese Klasse von `DataGridViewTextBoxCell` abzuleiten, um die Darstellung des Inhalts der Zelle für den Nicht-Editiermodus und verschiedene andere benötigte Methoden zu erben.
- Eine dritte Klasse implementiert schließlich den Spaltentyp und wird von `DataGridViewColumn` abgeleitet.

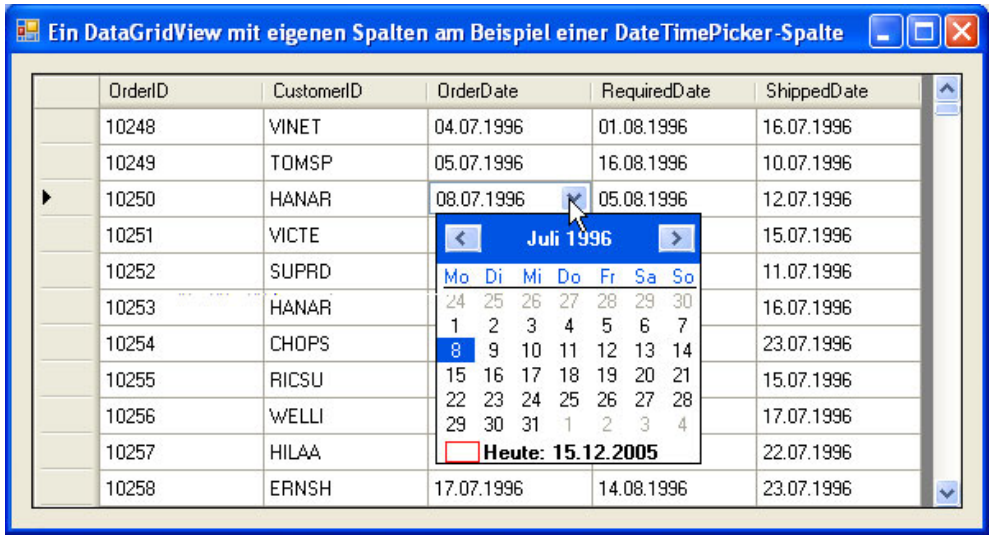


Abbildung 150: Ein DataGridView mit DateTimePicker-Spalten

Für unser Beispiel benötigen wir also zunächst eine Klasse für das Steuerelement. Ich implementiere diese Klasse öffentlich, damit ein Programmierer später für spezielle Anpassungen in den Ereignissen des `DataGridView` an das Steuerelement herankommt. Die Klasse wird von `DateTimePicker` abgeleitet und implementiert `IDataGridViewEditingControl`.

Zum Kompilieren dieser Klasse müssen Sie die Namensräume `System` und `System.Windows.Forms` importieren.

```
public class DataGridDateTimePicker : DateTimePicker,
    IDataGridViewEditingControl
{
   /* Gibt an, ob der im Steuerelement verwaltete Wert
      zwischenzeitlich geändert wurde */
   private bool valueChanged = false;
```

Listing 476: Beginn der Klasse für ein DateTimePicker-Steuerelement in einem DataGridView

```
/* Konstruktor. Setzt das Datumsformat auf das kurze Format. */
public DataGridDateTimePicker()
{
   this.Format = DateTimePickerFormat.Short;
}
```

Listing 476: Beginn der Klasse für ein DateTimePicker-Steuerelement in einem DataGridView (Forts.)

Die Schnittstelle `IDataGridViewEditingControl` besteht aus einigen Eigenschaften und Methoden, die Sie natürlich implementieren müssen.

Die Eigenschaft `EditingControlDataGridView` verwaltet eine Referenz auf das `DataGridView`-Steuerelement:

```
private DataGridView dataGridView;
public DataGridView EditingControlDataGridView
{
   get
   {
      return this.dataGridView;
   }

   set
   {
      this.dataGridView = value;
   }
}
```

Listing 477: Implementierung der EditingControlDataGridView-Eigenschaft

`EditingControlFormattedValue` verwaltet den formatierten Wert des Steuerelements. Das `DataGridDateTimePicker`-Steuerelement setzt hier das kurze Datumsformat ein:

```
public object EditingControlFormattedValue
{
   get
   {
      return this.Value.ToShortDateString();
   }

   set
   {
      if (value != null)
      {
```

Listing 478: Implementierung der EditingControlFormattedValue-Eigenschaft

```
            this.Value = DateTime.Parse(value.ToString());
        }
    }
}
```

Listing 478: Implementierung der EditingControlFormattedValue-Eigenschaft (Forts.)

Die Eigenschaft `EditingControlRowIndex` verwaltet den Index der Zeile, in der das Steuerelement angelegt ist (hier in einem privaten Feld):

```
private int rowIndex;
public int EditingControlRowIndex
{
    get
    {
        return this.rowIndex;
    }

    set
    {
        this.rowIndex = value;
    }
}
```

Listing 479: Implementierung der EditingControlRowIndex-Eigenschaft

Die Eigenschaft `EditingPanelCursor` legt den Mauscursor fest, der angezeigt wird, wenn die Maus sich zwar über der Zelle, aber nicht über dem Editier-Steuerelement befindet. Für unser Steuerelement kann hier der Cursor der Basisklasse verwendet werden:

```
public Cursor EditingPanelCursor
{
    get
    {
        return base.Cursor;
    }
}
```

Listing 480: Implementierung der EditingPanelCursor-Eigenschaft

Über die Eigenschaft `EditingControlValueChanged` erfährt das `DataGridView`-Steuerelement, ob der Wert, den das Zellen-Steuerelement verwaltet, geändert wurde.

```
public bool EditingControlValueChanged
{
   get
   {
      return this.valueChanged;
   }

   set
   {
      this.valueChanged = value;
   }
}
```

Listing 481: Implementierung der EditingControlValueChanged-Eigenschaft

Die Methode `GetEditingControlFormattedValue` liefert einfach nur den formatierten Wert des Steuerelements zurück (warum auch immer, schließlich reicht eigentlich die Eigenschaft `EditingControlFormattedValue`, die ich auch einfach hier verwende, aus):

```
public object GetEditingControlFormattedValue(
    DataGridViewDataErrorContexts context)
{
   return this.EditingControlFormattedValue;
}
```

Listing 482: Implementierung der GetEditingControlFormattedValue-Methode

In der Methode `ApplyCellStyleToEditingControl` wird der Stil der Zelle, den Sie über das übergebene Argument `dataGridViewCellStyle` erreichen, in das Steuerelement übertragen. Für unseren Fall werden die Schriftart, die Vorder- und die Hintergrundfarbe übergeben:

```
public void ApplyCellStyleToEditingControl(
    DataGridViewCellStyle dataGridViewCellStyle)
{
   this.Font = dataGridViewCellStyle.Font;
   this.CalendarForeColor = dataGridViewCellStyle.ForeColor;
   this.CalendarMonthBackground = dataGridViewCellStyle.BackColor;
}
```

Listing 483: Implementierung der ApplyCellStyleToEditingControl-Methode

Die Methode `EditingControlWantsInputKey` wird bei der Betätigung von Tasten aufgerufen. Die betätigten Tasten werden im Argument `keys` übergeben. Diese Methode muss `true` zurückgeben, falls das Steuerelement die betätigten Tasten selbst benötigt. In diesem Fall werden die Tasten nicht vom `DataGridView` ausgewertet. Im Argument `DataGridViewWantsInputKey` können

Sie abfragen, ob das DataGridView die betätigten Tasten überhaupt für sich in Anspruch nimmt. Da das DateTimePicker-Steuerelement die Positionierungs-Tasten zur Positionierung im aufgeklappten Kalender einsetzt, werden diese in unserer Klasse entsprechend abgefragt:

```
public bool EditingControlWantsInputKey(
    Keys keys, bool dataGridViewWantsInputKey)
{
   switch (keys & Keys.KeyCode)
   {
      // Das Steuerelement benötigt die hier aufgelisteten
      // Tasten selbst
      case Keys.Left:
      case Keys.Up:
      case Keys.Down:
      case Keys.Right:
      case Keys.Home:
      case Keys.End:
      case Keys.PageDown:
      case Keys.PageUp:
         return true;

      default:
         // Alle anderen Tasten werden nicht benötigt
         return false;
   }
}
```

Listing 484: Implementierung der EditingControlWantsInputKey-Methode

Die Methode PrepareEditingControlForEdit wird aufgerufen, wenn das Steuerelement in den Editiermodus umgeschaltet werden soll. Das Argument selectAll gibt an, ob der Inhalt komplett selektiert werden soll. Für unser Steuerelement ist hier keine Implementierung notwendig, da der DateTimePicker sich immer im Editiermodus befindet:

```
public void PrepareEditingControlForEdit(bool selectAll)
{
   // Für dieses Steuerelement ist hier keine Implementierung notwendig
}
```

Listing 485: Implementierung der PrepareEditingControlForEdit-Methode

RepositionEditingControlOnValueChange gibt an, ob das DataGridView-Steuerelement die Zelle neu positionieren (vergrößern oder verkleinern) soll, wenn der Wert des Editier-Steuerelements geändert wird. Hier sollten Sie true zurückgeben, wenn die Größe des Steuerelements sich dynamisch an den Inhalt anpasst. Da dies für unser Steuerelement nicht notwendig ist, wird false zurückgegeben.

```
   public bool RepositionEditingControlOnValueChange
   {
      get
      {
         return false;
      }
   }
```

Listing 486: Implementierung der RepositionEditingControlOnValueChange-Methode

Schließlich muss nur noch das private Feld `valueChanged` auf `true` gesetzt und die `NotifyCurrentCellDirty`-Methode des `DataGridView` aufgerufen werden, wenn der Wert des Steuerelements geändert wird. `valueChanged` wird in unserem Beispiel in der Eigenschaft `EditingControlValueChanged` abgefragt. `NotifyCurrentCellDirty` informiert das `DataGridView`-Steuerelement darüber, dass der Wert der Zelle geändert wurde. Die `DataGridDateTimePicker`-Klasse überschreibt zu diesem Zweck die geerbte `OnValueChanged`-Methode:

```
   protected override void OnValueChanged(EventArgs eventargs)
   {
      this.valueChanged = true;
      this.dataGridView.NotifyCurrentCellDirty(true);

      // Die geerbte Methode aufrufen,
      // um das ValueChanged-Ereignis aufzurufen
      base.OnValueChanged(eventargs);
   }
}
```

Listing 487: Überschreiben der OnValueChanged-Methode und Abschluss der DataGridDateTimePicker-Klasse

Die Klasse für die Zelle leite ich für unser Beispiel von der Klasse `DataGridViewTextBoxCell` ab. So kann ich auf die Logik zur Darstellung des Zellinhalts im Anzeigemodus und auf die Implementierung verschiedener notwendiger Methoden verzichten.

Im Konstruktor wird das Format der Zelle auf das kurze Datumsformat eingestellt:

```
public class DataGridViewDateTimePickerCell: DataGridViewTextBoxCell
{
   public DataGridViewDateTimePickerCell()
   {
      this.Style.Format = "d";
   }
```

Listing 488: Beginn der DataGridViewDateTimePickerCell-Klasse und Konstruktor

Die zu überschreibende Eigenschaft `EditType` gibt den Typ des Editier-Steuerelements zurück. `ValueType` gibt den Typ des im Editier-Steuerelement verwalteten Werts zurück. `DefaultNewRowValue` liefert einen Defaultwert für neue Zeilen. Die `DataGridViewDateTimePickerCell`-Klasse liefert hier das aktuelle Datum zurück:

```
/* Gibt den Typ des Editier-Steuerelements zurück */
public override Type EditType
{
   get
   {
      return typeof(DataGridDateTimePicker);
   }
}

/* Gibt den Typ des im Editier-Steuerelement verwalteten Werts zurück */
public override Type ValueType
{
   get
   {
      return typeof(DateTime);
   }
}

/* Gibt den Defaultwert für neue Zeilen zurück */
public override object DefaultNewRowValue
{
   get
   {
      return DateTime.Now;
   }
}
```

Listing 489: Die wichtigen Eigenschaften EditType, ValueType und DefaultNewRowValue

Die zu überschreibende Methode `InitializeEditingControl` initialisiert das Editier-Steuerelement. Im Argument `initialFormattedValue` wird der (formatierte) Wert übergeben, der zurzeit in der Zelle gespeichert wird. Das Argument `dataGridViewCellStyle` referenziert den Stil der Zelle. Das `DateTimePicker`-Beispiel referenziert das Editier-Steuerelement und übergibt den nach `DateTime` konvertierten Wert, falls dieser nicht `null` ist und dessen `ToString`-Methode keinen Leerstring ergibt. Im anderen Fall wird der minimal mögliche Wert des `DataGridDateTimePicker`-Steuerelements in das Steuerelement geschrieben. Schließlich wird noch die geerbte Methode aufgerufen. Dies besitzt für die `DataGridViewDateTimePickerCell`-Klasse zwar keine besondere Bedeutung. Allerdings erzeugt diese Methode die in der Basisklasse verwendete `TextBox`, auf die in anderen Methoden der Basisklasse zugegriffen wird. Verzichten Sie auf den Aufruf von `base.InitializeEditingControl`, resultiert dies in einer `NullReferenceException` (in

der privaten WireEditingControlEvents-Methode des DataGridView-Steuerelements, die verschiedene Ereignismethoden an die Ereignisse des aktuellen Editier-Steuerelements hängt) beim Umschalten in den Editiermodus.

```
   public override void InitializeEditingControl(int rowIndex,
       object initialFormattedValue,
       DataGridViewCellStyle dataGridViewCellStyle)
   {
      // Den Wert der Zelle an das Editier-Steuerelement übergeben
      DataGridDateTimePicker ctl =
          DataGridView.EditingControl as DataGridDateTimePicker;
      if (initialFormattedValue != null &&
         initialFormattedValue.ToString().Length > 0)
      {
         ctl.Value = Convert.ToDateTime(initialFormattedValue);
      }
      else
      {
         ctl.Value = ctl.MinDate;
      }

      // Die geerbte Methode aufrufen, da diese die TextBox der Basisklasse
      // erzeugt, die in anderen Methoden benötigt wird (der Nicht-Aufruf
      // der geerbten Methode führt ansonsten zu einer NullReferenceException
      // beim Umschalten in den Editiermodus)
      base.InitializeEditingControl(rowIndex, initialFormattedValue,
          dataGridViewCellStyle);
   }
}
```

Listing 490: Initialisierung des Editier-Steuerelements und Abschluss der DataGridViewDateTimePickerCell-Klasse

Die DataGridViewDateTimePickerColumn-Klasse, die die Spalte repräsentiert, ist schließlich recht einfach zu implementieren. Im Konstruktor wird die Basisklasse mit einer neuen Instanz der Klasse für die Zelle (hier: DataGridViewDateTimePickerCell) initialisiert:

```
public class DataGridViewDateTimePickerColumn : DataGridViewColumn
{
   public DataGridViewDateTimePickerColumn()
      : base(new DataGridViewDateTimePickerCell())
   {
   }
```

Listing 491: Beginn der DataGridViewDateTimePickerColumn-Klasse und Konstruktor

Dann muss noch die CellTemplate-Eigenschaft so überschrieben werden, dass diese beim Schreiben lediglich Instanzen der DataGridViewDateTimePickerCell- oder einer davon abgeleiteten Klasse zulässt:

```
   public override DataGridViewCell CellTemplate
   {
      get
      {
         return base.CellTemplate;
      }

      set
      {
         // Sicherstellen, dass das übergebene Objekt ein
         // DataGridViewDateTimePickerCell-Objekt ist
         if (value != null &&
            (value is DataGridViewDateTimePickerCell) == false)
         {
            throw new InvalidCastException("CellTemplate muss eine " +
               " Instanz der DataGridViewDateTimePickerCell-Klasse oder " +
               " einer davon abgeleiteten Klasse sein");
         }
         base.CellTemplate = value;
      }
   }
}
```

Listing 492: Die überschriebene CellTemplate-Eigenschaft und Abschluss der DataGridViewDateTimePickerColumn-Klasse

Wenn Sie diese Klassen in ein Projekt integrieren, können Sie die DataGridViewDateTimePickerColumn-Klasse sogar im Spalten-Designer eines DataGridView-Steuerelements verwenden (Abbildung 151).

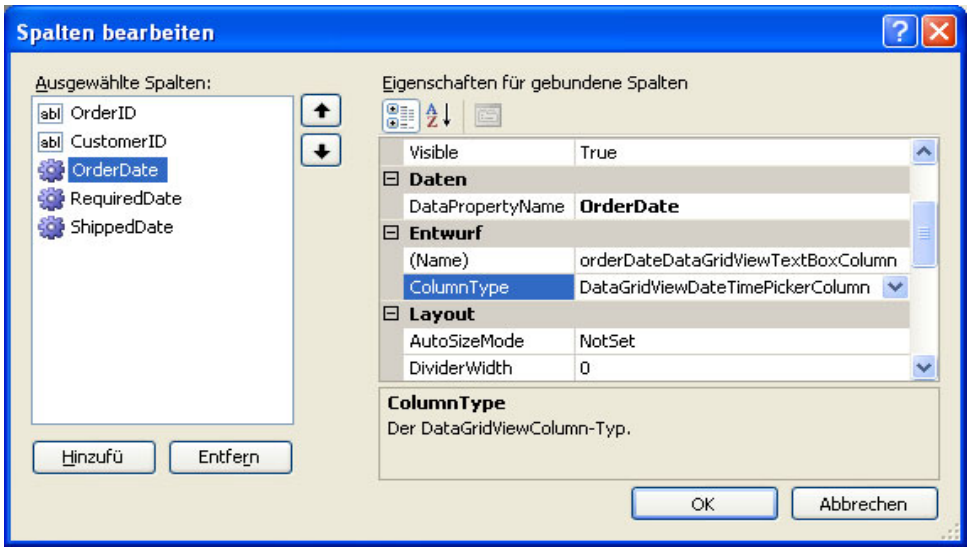


Abbildung 151: Zuweisung der DataGridViewDateTimePickerColumn-Klasse im Spalten-Designer eines DataGridView-Steuerelements

254 Feiertage im MonthCalendar-Steuerelement darstellen

Das `MonthCalendar`-Steuerelement bietet über die Eigenschaft `BoldedDates` die Möglichkeit, Tage hervorgehoben darzustellen. In diese Eigenschaft müssen Sie ein Array von `DateTime`-Instanzen schreiben, das die hervorzuhebenden Tage enthält. Die deutschen Feiertage können Sie über die Methode `GetGermanSpecialDays` aus dem Rezept 72 ermitteln. Diese müssen Sie lediglich in ein `DateTime`-Array kopieren und der `BoldedDates`-Auflistung zuweisen. Diese Zuweisung müssen Sie wiederholen, wenn das angezeigte Jahr wechselt. Außerdem wäre es schön, wenn der Benutzer den Namen des Feiertags in einem TooltTip angezeigt bekommen würde, wenn er mit der Maus über eines der hervorgehobenen Daten fährt.

Um das Ganze zu vereinfachen habe ich das Steuerelement `GermanMonthCalendar` entwickelt, das von `MonthCalendar` abgeleitet ist, und dieses um die Feiertags-Funktionalität erweitert. Kern dieser Klasse sind die Methoden `getHolidays` und `timer_Tick`. `getHolidays` wird immer dann aufgerufen, wenn das Datum geändert wurde (also einmal im Konstruktor und in der überschriebenen `OnDateChanged`-Methode). Diese Methode liest über die `GetGermanSpecialDays`-Methode aus dem Rezept 72 die deutschen Spezialtage ein, geht diese durch und schreibt die aus den Spezialtagen ermittelten Feiertage in die private Auflistung `holidays`. Danach wird der ebenfalls private Timer gestartet, der nach 100 Millisekunden sein `Tick`-Ereignis ausführt. In diesem Ereignis werden die Datumswerte der Feiertage in ein `DateTime`-Array kopiert und der `BoldedDates`-Eigenschaft zugewiesen. Die Verwendung des Timers ist ein kleiner Workaround: Ohne diese kurze Verzögerung würde das Steuerelement ansonsten bei einem Klick auf einen der Pfeile für den nächsten bzw. den vorherigen Monat nicht mehr registrieren, dass der Klick beendet wurde, und die Monate einfach weiter durchscrollen.

Um dem Anwender die Feiertage in einem `ToolTip` anzuzeigen überschreibt `GermanMonthCalendar` schließlich noch die `OnMouseMove`-Methode. In dieser Methode ermittelt das Steuerelement

über die geerbte HitTest-Methode das Datum, auf dem die Maus liegt. Dann geht sie die aktuellen Feiertage durch und überprüft, ob das Datum zu einem Feiertag gehört. Ist dies der Fall, wird der Name des Feiertags in dem ToolTip ausgegeben und die Methode beendet. Im anderen Fall wird der ToolTip einfach zurückgesetzt.

Zum Kompilieren dieser Klasse müssen Sie die Namensräume System, System.Windows.Forms importieren und eine Klasse DateUtils implementieren, die die Typen und Methoden aus dem Rezept 72 beinhaltet.

```
public class GermanMonthCalendar : MonthCalendar
{
   /* Verwaltet die aktuellen Feiertage */
   private DateUtils.GermanSpecialDays holidays;

   /* Timer für das zeitverzögerte Ermitteln der speziellen Tage */
   private Timer timer;

   /* ToolTip für die Anzeige der Feiertage */
   private ToolTip toolTip = new ToolTip();

   /* Konstruktor. Initialisiert den Timer und ermittelt die ersten
      Feiertage */
   public GermanMonthCalendar()
   {
      this.timer = new Timer();
      this.timer.Interval = 100;
      this.timer.Tick += new EventHandler(this.timer_Tick);
      this.getHolidays(this.SelectionStart.Year);
   }

   /* Ermittelt die speziellen Tage neu, wenn das Jahr wechselt */
   protected override void OnDateChanged(DateRangeEventArgs e)
   {
      base.OnDateChanged(e);

      // Feiertage ermitteln
      this.getHolidays(e.Start.Year);
   }

   /* Ermittelt die Feiertage */
   private void getHolidays(int year)
   {
      if (this.holidays == null || this.holidays.Year != year)
      {
         // Die speziellen Tage für das angezeigte Jahr ermitteln
         DateUtils.GermanSpecialDays specialDays =
```

Listing 493: Spezielle MonthCalendar-Klasse, die die deutschen Feiertage darstellt

```
            DateUtils.GetGermanSpecialDays(year);

         // Die Feiertage auslesen
         this.holidays = new DateUtils.GermanSpecialDays(year);
         foreach (DateUtils.GermanSpecialDay specialDay in
            specialDays.Values)
         {
            if (specialDay.IsHoliday)
            {
               this.holidays.Add(specialDay.Key, specialDay);
            }
         }

         // Timer starten, der etwas zeitverzögert die
         // Feiertage ermittelt und in BoldedDates schreibt
         this.timer.Start();
      }
   }

   /* Schreibt etwas zeitverzögert die Feiertage in die
      BoldedDates-Eigenschaft */
   private void timer_Tick(object sender, EventArgs e)
   {
      // Timer beenden
      this.timer.Stop();

      // Die Tage der Feiertage in ein DateTime-Array kopieren und dieses der
      // BoldedDates-Eigenschaft zuweisen
      DateTime[] specialDates = new DateTime[this.holidays.Count];
      int index = -1;
      foreach (DateUtils.GermanSpecialDay specialDay in this.holidays.Values)
      {
         index++;
         specialDates[index] = specialDay.Date;
      }
      this.BoldedDates = specialDates;
   }

   /* Zeigt einen ToolTip an, wenn sich die Maus auf einem
      Feiertag befindet */
   protected override void OnMouseMove(MouseEventArgs e)
   {
      base.OnMouseMove(e);

      if (this.holidays != null)
```

Listing 493: Spezielle MonthCalendar-Klasse, die die deutschen Feiertage darstellt (Forts.)

```
        {
            DateTime date = this.HitTest(e.X, e.Y).Time;
            foreach (DateUtils.GermanSpecialDay holiday in this.holidays.Values)
            {
                if (holiday.Date.Date == date.Date)
                {
                    this.toolTip.SetToolTip(this, holiday.Name);
                    return;
                }
            }
            // ToolTip löschen
            this.toolTip.SetToolTip(this, null);
        }
    }
}
```

Listing 493: Spezielle MonthCalendar-Klasse, die die deutschen Feiertage darstellt (Forts.)

Abbildung 152 zeigt ein einfaches Windows-Programm, das dieses Steuerelement einsetzt.

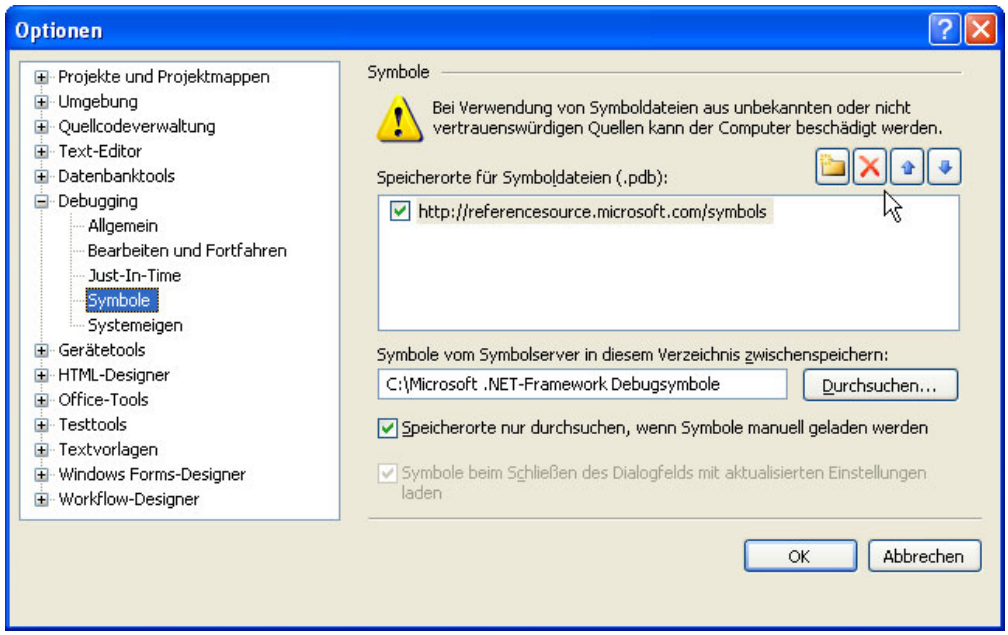


Abbildung 152: Anzeige der deutschen Feiertage im GermanMonthCalendar-Steuerelement

255 Drag&Drop von Dateien und Ordnern

Für Anwendungen, die Eingaben von Datei- oder Ordnernamen erwarten, ist es sinnvoll, Drag&Drop vom Explorer aus zu ermöglichen. Dieses Problem ist einfach gelöst, aber nicht unbedingt so klar, dass das Rezept hier sinnlos wäre ☺.

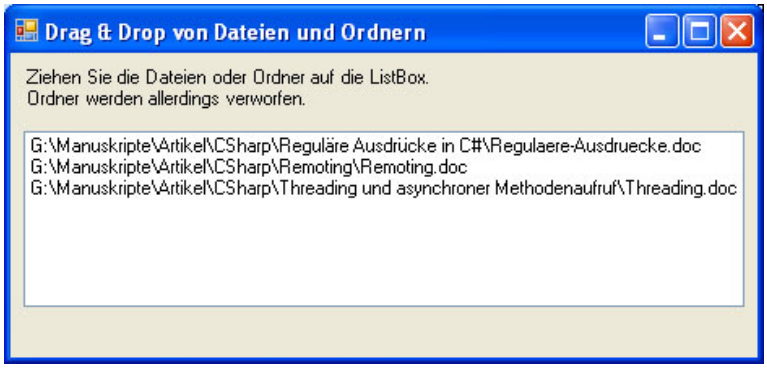


Abbildung 153: Beispielanwendung für das Drag&Drop von Dateien und Ordnern

In der Beispielanwendung sollen Dateien und Ordner auf die `ListBox` mit Namen *fileListBox* gezogen werden können. Dazu stellen Sie zunächst die Eigenschaft `AllowDrop` der `ListBox` auf `true`.

Im Ereignis `DragEnter` der `ListBox` überprüfen Sie über die `GetDataPresent`-Methode des im `DragEventArgs`-Argument übergebenen `IDataObject`-Objekts, ob Dateien oder Ordner gezogen werden. Dazu übergeben Sie die Konstante `DataFormats.FileDrop`. In diesem Fall setzen Sie den Effekt auf `DragDropEffects.Copy`, da Sie im Prinzip nur das »Kopieren« der Dateien und nicht das Verschieben zulassen (das Sie natürlich implementieren müssten):

```
private void fileListBox_DragEnter(object sender, DragEventArgs e)
{
   // Überprüfen, ob Dateien oder Ordner gezogen werden
   if (e.Data.GetDataPresent(DataFormats.FileDrop))
   {
      e.Effect = DragDropEffects.Copy;
   }
   else
   {
      e.Effect = DragDropEffects.None;
   }
}
```

Listing 494: Setzen des Drag&Drop-Effekts

Im Ereignis `DragDrop` werten Sie nun die gezogenen Daten aus. Dazu rufen Sie die `GetData`-Methode des mit dem `DragEventArgs`-Objekt in dessen `Data`-Eigenschaft übergebenen `IDataObject`-Objekts aus. Am ersten Argument können Sie ein `Type`-Objekt übergeben (was für gezogene Dateien und Ordner aber nicht existiert). Alternativ können Sie auch einen String übergeben, der den Typ kennzeichnet. Und hier können Sie direkt `DataFormats.FileDrop` übergeben, weil diese Konstante den passenden String (”FileDrop”) verwaltet. Am zweiten Argument definieren Sie mit `false`, dass die Daten nicht in das angegebene Format konvertiert werden sollen, weil diese ja bereits in diesem Format vorliegen.

Sie erhalten dann ein String-Array mit den Datei- und/oder Ordnernamen. Alles Weitere bleibt dann Ihnen überlassen. Das Beispiel ermittelt über ein `FileInfo`-Objekt (aus dem Namensraum `System.IO`), ob es sich um eine Datei handelt, und fügt dieses Objekt dann einfach an die Liste an. Da die `ToString`-Methode eines `FileInfo`-Objekts den vollen Dateinamen zurückgibt, wird dieser in der Liste angezeigt. Der Vorteil dieser Vorgehensweise ist, dass Sie bei der Auswertung der Liste über die referenzierten `FileInfo`-Objekte die vollen Informationen zu den Dateien zur Verfügung haben.

```
private void fileListBox_DragDrop(object sender, DragEventArgs e)
{
   // Dateien aus den gezogenen Daten auslesen
   string[] filesNames = (string[])e.Data.GetData(DataFormats.FileDrop,
      false);
   foreach (string fileName in filesNames)
   {
      // FileInfo-Objekt erzeugen
      FileInfo fi = new FileInfo(fileName);
      if (fi.Exists)
      {
         // Wenn es sich nicht um einen Ordner handelt: FileInfo-Objekt der
         // Liste anfügen
         this.fileListBox.Items.Add(fi);
      }
   }
}
```

Listing 495: Auslesen der gezogenen Dateinamen

256 Ein Ordner-Dialog

Das .NET Framework bietet über die Klasse `FolderBrowserDialog` (aus dem Namensraum `System.Windows.Forms`) die Möglichkeit, einen Ordner-Dialog anzuzeigen. Über die Eigenschaft `SelectedPath` können Sie den voreingestellten Ordner einstellen. Neben normalen Ordnerangaben können Sie auch die speziellen Windows-Ordner über die Aufzählung `Environment.SpecialFolder` angeben. Die Eigenschaft `ShowNewFolderButton` bestimmt, ob der Schalter zum Erstellen eines neuen Ordners sichtbar ist. Über die Eigenschaft `Description` können Sie einen Text angeben, der über der Ordnerliste erscheint. Die `ShowDialog`-Methode öffnet den Dialog dann wie gewohnt:

```
FolderBrowserDialog fbd = new FolderBrowserDialog();
fbd.SelectedPath = "C:\\";
// fbd.RootFolder = Environment.SpecialFolder.Personal;
fbd.ShowNewFolderButton = true;
fbd.Description = "Wählen Sie einen Ordner aus:";
```

Listing 496: Ein Ordner-Dialog über die Klasse FolderBrowserDialog

```
if (fbd.ShowDialog() == DialogResult.OK)
{
   Console.WriteLine(fbd.SelectedPath);
}
```

Listing 496: Ein Ordner-Dialog über die Klasse FolderBrowserDialog (Forts.)

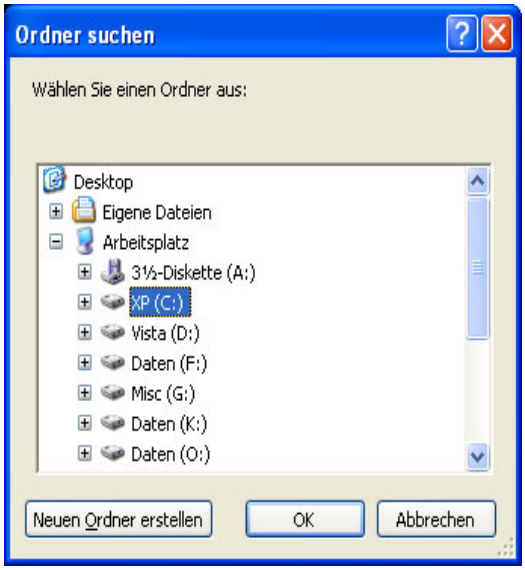


Abbildung 154: Der geöffnete Ordner-Dialog

257 In einem Nicht-Tastatur-Ereignis herausfinden, ob eine bestimmte Taste betätigt ist

In den Ereignissen `KeyPress`, `KeyDown` und `KeyUp` erhalten Sie über das Ereignisargument-Objekt eine Information über die aktuell betätigten Tasten. Manchmal benötigen Sie aber auch an anderer Stelle im Programm eine Information darüber, ob bestimmte Tasten betätigt sind. In Zeichen-Programmen ist es zum Beispiel üblich, dass beim Zeichnen von Kreisen zunächst eine Ellipse gezeichnet wird und nur dann ein Kreis, wenn der Benutzer gleichzeitig die Shift-Taste betätigt.

.NET hilft Ihnen hier nur bedingt weiter. Über die Eigenschaft `ModifierKeys` der `Control`-Klasse (aus dem Namensraum `System.Windows.Forms`) erhalten Sie eine Information darüber, welche der Modifizier-Tasten (Shift, Steuerung, Alt) betätigt sind. Für die meisten Anwendungen dürfte das bereits ausreichen. Wenn Sie aber herausfinden wollen, ob eine andere Taste betätigt ist, müssen Sie auf die Windows-API-Funktion `GetAsyncKeyState` zurückgreifen.

Diese Funktion wird folgendermaßen deklariert:

```
[DllImport("user32.dll")]
static extern short GetAsyncKeyState(Keys key);
```

`GetAsyncKeyState` gibt den derzeitig aktuellen Status einer Taste zurück (die ebenfalls verfügbare Funktion `GetKeyState` ist für die Reaktion auf eine Tastatur-Nachricht gedacht und gibt den Zustand der Taste zum Zeitpunkt dieser Nachricht zurück). Laut der Dokumentation setzt `GetAsyncKeyState` das signifikanteste Bit des Rückgabewerts auf 1, wenn die übergebene Taste gerade betätigt ist. Im am wenigsten signifikanten Bit erhalten Sie eine Information darüber, ob der Zustand der Taste seit dem letzten Aufruf der Funktion geändert wurde. Diese Information sollten Sie laut der Dokumentation aber nicht verwenden, da diese in bestimmten Situationen nicht verfügbar ist.

Ob eine Taste gerade betätigt ist, können Sie nun einfach darüber ermitteln, ob der zurückgegebene Wert negativ ist, denn dann ist in einem `short`-Wert das signifikanteste Bit (das Vorzeichenbit) gesetzt.

Das folgende Beispiel fragt `Control.ModifierKeys` ab um in einem `MouseDown`-Ereignis herauszufinden, ob die Modifizier-Tasten betätigt sind. Um herauszufinden, ob die Tasten F1, F2 oder F3 betätigt sind, setzt dieses Beispiel die `GetAsyncKeyState`-Funktion ein.

Zum Kompilieren des Beispiels benötigen Sie ein Formular mit Namen *StartForm*, dessen `MouseDown`-Ereignis Sie mit der Methode `StartForm_MouseDown` verknüpfen. Außerdem müssen Sie die Namensräume `System`, `System.Windows.Forms` und `System.Runtime.InteropServices` importieren.


```
[DllImport("user32.dll")]
static extern short GetAsyncKeyState(Keys key);

private void StartForm_MouseDown(object sender, MouseEventArgs e)
{
   string info = null;
   // Auf die Steuerung-Taste abfragen
   if ((Control.ModifierKeys & Keys.Control) > 0)
   {
      info = "Die Strg-Taste ist betätigt";
   }
   // Auf die Shift-Taste abfragen
   if ((Control.ModifierKeys & Keys.Shift) > 0)
   {
      if (info != null) info += Environment.NewLine;
      info += "Die Shift-Taste ist betätigt";
   }
   // Auf die Alt-Taste abfragen
   if ((Control.ModifierKeys & Keys.Alt) > 0)
   {
      if (info != null) info += Environment.NewLine;
      info += "Die Alt-Taste ist betätigt";
   }
```

Listing 497: In einem MouseDown-Ereignis ermitteln, ob bestimmte Tasten betätigt sind

```
   // Auf die ersten drei Funktionstasten abfragen
   if (GetAsyncKeyState(Keys.F1) < 0)
   {
      if (info != null) info += Environment.NewLine;
      info += "Die F1-Taste ist betätigt";
   }
   if (GetAsyncKeyState(Keys.F2) < 0)
   {
      if (info != null) info += Environment.NewLine;
      info += "Die F2-Taste ist betätigt";
   }
   if (GetAsyncKeyState(Keys.F3) < 0)
   {
      if (info != null) info += Environment.NewLine;
      info += "Die F3-Taste ist betätigt";
   }

   if (info != null)
   {
      MessageBox.Show(info, Application.ProductName, MessageBoxButtons.OK,
         MessageBoxIcon.Information);
   }
   else
   {
      MessageBox.Show("Keine Taste ist betätigt", Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Information);
   }
}
```

Listing 497: In einem MouseDown-Ereignis ermitteln, ob bestimmte Tasten betätigt sind (Forts.)

258 In einem eigenen Steuerelement verhindern, dass die Cursor-Tasten einen Fokuswechsel bewirken

Wenn Sie ein eigenes Steuerelement implementieren, das seine Darstellung in der überschriebenen `OnPaint`-Methode selbst zeichnet, und dieses im Konstruktor als selektierbar kennzeichnen (`this.SetStyle(ControlStyles.Selectable, true)`), wollen Sie unter Umständen die Cursortasten als Eingabetasten auswerten. Dies ist zum Beispiel in einem Steuerelement der Fall, das eine Liste von Daten anzeigt, in der der Benutzer über die Cursortasten navigieren kann.

Dummerweise wertet .NET die Cursortasten aber so aus, dass diese einen Fokuswechsel auf das nächste bzw. vorherige Steuerelement bewirken. Wollen Sie dies verhindern, müssen Sie diese Tasten als Eingabetasten kennzeichnen. Dazu überschreiben Sie die Methode `IsInputKey` und geben `true` zurück, wenn es sich um eine der genannten Tasten handelt. In allen anderen Fällen geben Sie den Wert zurück, den die geerbte Methode liefert:

```
protected override bool IsInputKey(Keys key)
{
   switch(key)
   {
      case Keys.Up:
      case Keys.Down:
      case Keys.Right:
      case Keys.Left:
         return true;
   }
   return base.IsInputKey(key);
}
```

Listing 498: Überschreiben der IsInputKey-Methode um einen Fokuswechsel durch die Cursortasten zu vermeiden

Wenn Sie die `IsInputKey`-Methode auf diese Weise überschreiben, kann der Benutzer trotzdem noch über eine Kombination aus der Shift- und einer Cursortaste zum nächsten bzw. vorherigen Steuerelement wechseln. Wollen Sie auch dies verhindern, müssen Sie diese Kombinationen ebenfalls abfangen:


```
protected override bool IsInputKey(Keys key)
{
   switch(key)
   {
      case Keys.Up:
      case Keys.Down:
      case Keys.Right:
      case Keys.Left:
      case Keys.Up | Keys.Shift:
      case Keys.Down | Keys.Shift:
      case Keys.Right | Keys.Shift:
      case Keys.Left | Keys.Shift:
         return true;
   }
   return base.IsInputKey(key);
}
```

Listing 499: Überschreiben der IsInputKey-Methode um einen Fokuswechsel durch die Cursortasten auch in einer Kombination mit der Shift-Taste zu vermeiden

Hinweis

Das Beispiel zu diesem Rezept implementiert ein (sehr) einfaches Steuerelement zur Auswahl einer Farbe, das diese Technik demonstriert.

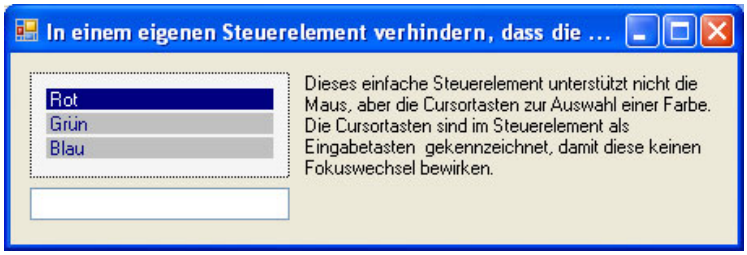


Abbildung 155: Beispielanwendung mit einem einfachen Steuerelement zur Auswahl einer Farbe

259 Hooking – Eine MessageBox mit definierten Schalterbeschriftungen versehen

Die in .NET über die Klasse `MessageBox` implementierte Standard-Windows-Message-Box besitzt leider nur vordefinierte Schalterbeschriftungen. In vielen Fällen passen diese aber nicht zu der jeweiligen Anwendung. Wenn Sie den Anwender beispielsweise fragen wollen, ob er eine Protokolldatei neu erzeugen oder an eine vorhandene Datei anfügen will, können Sie mit den Default-Schaltern (JA – NEIN, JA – NEIN – ABBRECHEN, ABBRECHEN – IGNORIEREN – WIEDERHOLEN etc.) nicht viel anfangen. Ideal wäre eine `MessageBox` wie in Abbildung 156.



Abbildung 156: MessageBox mit speziellen Schalterbeschriftungen

Um dieses Problem zu lösen können Sie ein Windows-Formular erzeugen, das den Text, die Schalter und das Icon je nach Aufruf einer *Show*-Methode erzeugt. Das macht aber sehr viel Arbeit, wenn Sie das Formular möglichst flexibel einsetzbar machen wollen.

Einfacher ist es, die originale Windows-`MessageBox` zu verändern. Über die API-Funktion `SetDlgItemText` können Sie die Beschriftungen einzelner Schalter im Fenster eines Dialogs (also auch einer `MessageBox`) ändern. Dazu übergeben Sie den Windows-Handle des Dialogs, eine Id für den Schalter und den neuen Text. Die Ids der Schalter einer `MessageBox` sind in Windows festgelegt. Der Wert 1 steht z. B. für einen OK-Schalter, 2 steht für einen Abbrechen-Schalter.

Das Problem ist nur, dass die `MessageBox` modal ist und Sie folglich nach dem Öffnen nicht an das Fenster herankommen. Ein weiteres Problem wäre auch im anderen Fall, dass Sie den Windows-Handle des `MessageBox`-Fensters herausfinden müssen.

Die Lösung dieser Probleme ist das so genannte *Hooking*. Hooking bedeutet, dass Sie eine spezielle (Hook-)Funktion so installieren, dass diese bestimmte Windows-Nachrichten abfängt. Wie Sie ja wahrscheinlich bereits wissen, basiert die Interaktion zwischen dem Anwender und

den Programm-Fenstern in Windows auf Nachrichten, die Windows bei allen möglichen Ereignissen an die Fenster sendet. Fenster verarbeiten diese Nachrichten und reagieren entsprechend darauf. In modernen Programmiersystemen werden Windows-Nachrichten in Ereignisse umgewandelt. Wenn Sie z. B. für das Ereignis `MouseMove` eines Formulars eine Behandlungsmethode schreiben, reagieren Sie in Wirklichkeit auf die vom Fenster empfangene Windows-Nachricht `WM_MOUSEMOVE`.

Über einen Hook können Sie auf Windows-Nachrichten auch dann reagieren, wenn Sie ein Fenster nicht selbst programmiert haben. Sie installieren dazu über die API-Funktion `SetWindowsHookEx` eine spezielle Hook-Funktion. Dabei definieren Sie die Art des Hooks über festgelegte API-Konstanten. Ein Hook des Typs `WH_KEYBOARD` reagiert z. B. auf alle Tastatur-Nachrichten (womit Sie einen globalen Tastatur-Handler entwickeln könnten). Die verfügbaren Hook-Typkonstanten sind in der API-Dokumentation zu `SetWindowsHookEx` beschrieben.

Für unseren Zweck eignet sich ein *CBT*-Hook. Dieser primär für »Computer Based Training«-Programme entwickelte Hook wird bei verschiedenen Nachrichten wie dem Erstellen, Aktivieren oder Zerstören eines Fensters aufgerufen. Da das Aktivieren eines Fensters genau das ist, was wir benötigen, verwende ich einen solchen Hook. Innerhalb der Hook-Funktion erhalten Sie über das Argument `code` eine Information über die Unterart des Hook. Die Konstante `HCBT_ACTIVATE` steht für Nachrichten, die beim Aktivieren eines Fensters aufgerufen werden. Aus dem Argument `wParam` können Sie dann den Handle des gerade aktivierten Fensters auslesen um damit Windows-API-Funktionen wie `SetDlgItemText` aufzurufen.

Hooks können Sie global (für alle Windows-Nachrichten) oder nur für einen Thread (den Haupt-Thread einer Anwendung) installieren. Globale Hooks können das System überwachen, sind aber recht gefährlich, wenn innerhalb der Hook-Funktion ein modaler Dialog aufgerufen wird (der das System dann lahmlegt) oder ein Fehler auftritt (der das System u. U. abschießt). Für unseren Zweck reicht aber ein weniger gefährlicher Thread-lokaler Hook. Um diesen zu erzeugen übergeben Sie `SetWindowsHookEx` im letzten Argument die Id des aktuellen Threads (die Sie über `System.AppDomain.GetCurrentThreadId` auslesen).

In der Hook-Funktion sollten Sie dafür Sorge tragen, dass Windows nach Ihrem Hook einen eventuell vorhandenen nächsten abarbeiten kann, was Sie durch den Aufruf der Funktion `CallNextHookEx` erreichen.

Ein einfaches Beispiel

Als erstes einfaches Beispiel erzeugt die folgende Konsolenanwendung eine `MessageBox` mit drei benutzerdefinierten Schaltern. Zunächst benötigen Sie eine Variable, die den Handle des Hooks verwaltet:

```
class Hooking
{
   private static IntPtr hHook;
```

Eine Aufzählung verwaltet die Konstanten der verschiedenen Hook-Typen (obwohl hier nur ein CBT-Hook verwendet wird):

```
   public enum HookType: int
   {
      WH_JOURNALRECORD = 0,
      WH_JOURNALPLAYBACK = 1,
```

```
      WH_KEYBOARD = 2,
      WH_GETMESSAGE = 3,
      WH_CALLWNDPROC = 4,
      WH_CBT = 5,
      WH_SYSMSGFILTER = 6,
      WH_MOUSE = 7,
      WH_HARDWARE = 8,
      WH_DEBUG = 9,
      WH_SHELL = 10,
      WH_FOREGROUNDIDLE = 11,
      WH_CALLWNDPROCRET = 12,
      WH_KEYBOARD_LL = 13,
      WH_MOUSE_LL = 14
   }
```

Für CBT-Hooks benötigen Sie eine weitere Aufzählung, die die Nachrichten spezifiziert, bei denen ein solcher Hook aufgerufen werden kann:

```
   private enum CBTHookAction: int
   {
      HCBT_MOVESIZE = 0,
      HCBT_MINMAX = 1,
      HCBT_QS = 2,
      HCBT_CREATEWND = 3,
      HCBT_DESTROYWND = 4,
      HCBT_ACTIVATE = 5,
      HCBT_CLICKSKIPPED = 6,
      HCBT_KEYSKIPPED = 7,
      HCBT_SYSCOMMAND = 8,
      HCBT_SETFOCUS = 9
   }
```

Zur Veränderung der Beschriftungen der `MessageBox`-Schalter über `SetDlgItemText` werden die einzelnen Schaltertypen über festgelegte Konstanten definiert, die Sie ebenfalls in einer Aufzählung deklarieren sollten:

```
  private enum ButtonType
   {
      IDOK = 1,
      IDCANCEL = 2,
      IDABORT = 3,
      IDRETRY = 4,
      IDIGNORE = 5,
      IDYES = 6,
      IDNO = 7
   }
```

Da Sie in echten .NET-Programmen keine Funktionsadressen direkt übergeben können, benötigen Sie einen Delegaten für die Hook-Funktion. Die Signatur ist dabei festgelegt, wobei Sie das eigentlich als `int` deklarierte erste Argument zur Vereinfachung auch mit dem kompatiblen Aufzählungstyp deklarieren können:

```
   private delegate int HookProcDelegate(CBTHookAction nCode,
      IntPtr wParam, IntPtr lParam);
```

Nun folgt die Deklaration der benötigten API-Funktionen:

```
[DllImport("user32.dll", CharSet=CharSet.Auto)]
private static extern IntPtr SetWindowsHookEx(HookType idHook,
   HookProcDelegate lpfn, IntPtr hMod, int dwThreadId);

[DllImport("user32.dll")]
private static extern int CallNextHookEx(IntPtr hhook,
   CBTHookAction code, IntPtr wParam, IntPtr lParam);

[DllImport("user32.dll")]
private static extern int UnhookWindowsHookEx(IntPtr hhook);

[DllImport("User32")]
static extern private bool SetDlgItemText(IntPtr hWnd,
   int nIDDlgItem, string lpString);
```

Die Hook-Funktion, die die eigentliche Arbeit übernimmt, muss dann der im Delegaten festgelegten Signatur entsprechen:

```
private static int HookProc(CBTHookAction code, IntPtr wParam,
   IntPtr lParam)
{
   if (code == CBTHookAction.HCBT_ACTIVATE)
   {
      // Wenn das Fenster gerade aktiviert wird:
      // Schalter neu beschriften
      SetDlgItemText(wParam, (int)ButtonType.IDYES, "Jau");
      SetDlgItemText(wParam, (int)ButtonType.IDNO,  "Nö");
      SetDlgItemText(wParam, (int)ButtonType.IDCANCEL,
         "Vielleicht");
   }

   // Zum nächsten Hook in der Hook-Kette wechseln
   return CallNextHookEx(hHook, code, wParam, lParam);
}
```

Die Hook-Funktion vergleicht zunächst, ob der CBT-Hook für das Aktivieren eines Fensters aufgerufen wurde. Da der Hook weiter unten nur für den aktuellen Thread installiert wird, kann die Funktion sicher sein, dass es sich dabei um das Fenster der `MessageBox` handelt. Über die Funktion `SetDlgItemText` wird dann für jeden der Schalter ein neuer Text definiert. Der Handle des `MessageBox`-Fensters wird dabei aus dem Argument `wParam` ausgelesen.

Wie bereits gesagt ist der Aufruf von `CallNextHookEx` zur Verzweigung in den nächsten Hook sehr wichtig. Dabei müssen Sie auch den Rückgabewert dieser Funktion, der vom verwendeten Hook abhängt, in der Hook-Funktion selbst zurückgeben. Nur so sichern Sie ab, dass ein Hook ohne die Gefahr eines Absturzes funktioniert.

In der `Main`-Funktion der Anwendung installieren Sie nun den Hook:

```
[STAThread]
static void Main(string[] args)
{
```

```
        // Hook installieren
        hHook = SetWindowsHookEx(HookType.WH_CBT,
           new HookProcDelegate(HookProc), IntPtr.Zero,
           (int)AppDomain.GetCurrentThreadId());
```

Die Installation erfolgt so, dass ein CBT-Hook erzeugt wird. Für die Hook-Funktion wird ein neuer Delegat erzeugt, der am zweiten Argument übergeben wird. Das dritte Argument wird in API-Programmen verwendet, wenn die Hook-Funktion innerhalb einer DLL gespeichert ist, und muss in unserem Fall auf `IntPtr.Zero` gesetzt werden. Am vierten Argument übergebe ich die Id des aktuellen Thread, damit der Hook nur für diesen installiert wird.

Hinweis

`AppDomain.GetCurrentThreadId` ist zwar ab dem .NET Framework 2.0 als obsolet gekennzeichnet, der vorgeschlagene Ersatz `System.Threading.Thread.CurrentThread.ManagedThreadId` liefert aber nicht dieselbe Id wie `AppDomain.GetCurrentThreadId` und das Hooking funktioniert damit nicht.

Dann kann die Standard-Message-Box so aufgerufen werden, dass diese auch drei Schalter besitzt:

```
        MessageBox.Show("C# ist doch cool, oder?",
           Application.ProductName, MessageBoxButtons.YesNoCancel,
           MessageBoxIcon.Question);
```

Schließlich müssen Sie den Hook auf jeden Fall wieder deinstallieren:

```
        UnhookWindowsHookEx(hHook);
    }
}
```

Und das Ergebnis ist überzeugend (Abbildung 157).



Abbildung 157: Das Ergebnis des Hookings der MessageBox

Eine Klasse für eine MessageBox mit benutzerdefinierten Schalterbeschriftungen

Um die Hooking-Komplexität zu verstecken und die Erzeugung einer `MessageBox` mit eigenen Schalterbeschriftungen zu erleichtern, habe ich eine Klasse `ExtMessageBox` entwickelt. Die `Show`-Methode dieser Klasse arbeitet ähnlich der gleichnamigen Methode der `MessageBox`-Klasse, nur dass diese Methode (aus Vereinfachungsgründen) nicht mehrfach überladen ist und an Stelle des `buttons`-Arguments drei Strings für die einzelnen Schalterbeschriftungen erwartet. Übergeben Sie an Stelle einer Beschriftung `null`, wird der entsprechende Schalter nicht dargestellt. Die Rückgabe der Methode ist ein Wert vom neuen Typ `ExtDialogResult`, der mit seinen Konstanten `FirstButton`, `SecondButton` und `ThirdButton` den Schalter definiert, der betätigt wurde.

Show arbeitet auf der Basis des vorhergehenden Beispiels. Abhängig von den übergebenen Schalterbeschriftungen erzeugt diese Methode eine MessageBox mit einem Ok-, einem Ja- und einem Nein- oder einem Ja-, einem Nein- und einem Abbrechen-Schalter. Da die übergebenen Beschriftungen in privaten Eigenschaften gespeichert werden, kann die Hook-Funktion ermitteln, welche Schalter die MessageBox besitzt, um diese gezielt zu verändern. Nach dem Aufruf der MessageBox vergleicht Show die Rückgabe auch wieder abhängig von den definierten Beschriftungen mit den DialogResult-Konstanten und gibt einen entsprechenden ExtDialogResult-Wert zurück.

Show erfordert den Import der Namensräume System, System.Windows.Forms und System.Runtime.InteropServices.

```
/* Aufzählung für die Rückgabe der Show-Methode der ExtMessageBox-Klasse */
public enum ExtDialogResult
{
   /* Der erste Schalter wurde betätigt */
   FirstButton,

   /* Der zweite Schalter wurde betätigt */
   SecondButton,

   /* Der dritte Schalter wurde betätigt */
   ThirdButton
}

/* Erweiterte MessageBox, die die Definition der Beschriftungen
   der Schalter erlaubt. */
public class ExtMessageBox
{
   /* Feld für den Hook-Handle */
   private static IntPtr hHook;

   /* Verwaltet die Beschriftung des ersten Schalters */
   private static string button1Caption;

   /* Verwaltet die Beschriftung des zweiten Schalters */
   private static string button2Caption;
   /* Verwaltet die Beschriftung des dritten Schalters */
   private static string button3Caption;

   /* Aufzählung für die verschiedenen Hook-Typen  */
   private enum HookType : int
   {
      WH_JOURNALRECORD = 0,
      WH_JOURNALPLAYBACK = 1,
```

Listing 500: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren

```
    WH_KEYBOARD = 2,
    WH_GETMESSAGE = 3,
    WH_CALLWNDPROC = 4,
    WH_CBT = 5,
    WH_SYSMSGFILTER = 6,
    WH_MOUSE = 7,
    WH_HARDWARE = 8,
    WH_DEBUG = 9,
    WH_SHELL = 10,
    WH_FOREGROUNDIDLE = 11,
    WH_CALLWNDPROCRET = 12,
    WH_KEYBOARD_LL = 13,
    WH_MOUSE_LL = 14
}

/* Aufzählung für die Aktionen, bei denen ein CBT-Hook aufgerufen
   werden kann */
private enum CBTHookAction : int
{
    HCBT_MOVESIZE = 0,
    HCBT_MINMAX = 1,
    HCBT_QS = 2,
    HCBT_CREATEWND = 3,
    HCBT_DESTROYWND = 4,
    HCBT_ACTIVATE = 5,
    HCBT_CLICKSKIPPED = 6,
    HCBT_KEYSKIPPED = 7,
    HCBT_SYSCOMMAND = 8,
    HCBT_SETFOCUS = 9
}

/* Aufzählung für die MessageBox-Schaltertypen  */
private enum ButtonType
{
    IDOK = 1,
    IDCANCEL = 2,
    IDABORT = 3,
    IDRETRY = 4,
    IDIGNORE = 5,
    IDYES = 6,
    IDNO = 7
}
/* Delegat für die Hook-Funktion */
private delegate int HookProcDelegate(CBTHookAction nCode,
    IntPtr wParam, IntPtr lParam);
```

Listing 500: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

```
#region Deklaration der benötigten API-Funktionen

[DllImport("user32.dll", CharSet = CharSet.Auto)]
private static extern IntPtr SetWindowsHookEx(HookType idHook,
   HookProcDelegate lpfn, IntPtr hMod, int dwThreadId);

[DllImport("user32.dll")]
private static extern int CallNextHookEx(IntPtr hhook,
   CBTHookAction code, IntPtr wParam, IntPtr lParam);

[DllImport("user32.dll")]
private static extern int UnhookWindowsHookEx(IntPtr hhook);

[DllImport("User32")]
static extern private bool SetDlgItemText(IntPtr hWnd,
   int nIDDlgItem, string lpString);

#endregion

/* Die Hook-Funktion  */
private static int HookProc(CBTHookAction code, IntPtr wParam,
   IntPtr lParam)
{
   if (code < 0)
   {
      // Wenn keine Hook-Aktion übergeben wurde, Windows
      // direkt zum nächsten Hook verzweigen lassen
      return CallNextHookEx(hHook, code, wParam, lParam);
   }

   if (code == CBTHookAction.HCBT_ACTIVATE)
   {
      // Wenn das Fenster gerade aktiviert wird:
      // Schalter neu beschriften
      if (button3Caption != null)
      {
         SetDlgItemText(wParam, (int)ButtonType.IDYES,
            button1Caption);
         SetDlgItemText(wParam, (int)ButtonType.IDNO,
            button2Caption);
         SetDlgItemText(wParam, (int)ButtonType.IDCANCEL,
            button3Caption);
      }
      else if (button2Caption != null)
      {
```

Listing 500: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

```
            SetDlgItemText(wParam, (int)ButtonType.IDYES,
               button1Caption);
            SetDlgItemText(wParam, (int)ButtonType.IDNO,
               button2Caption);
         }
         else
         {
            SetDlgItemText(wParam, (int)ButtonType.IDOK,
               button1Caption);
         }
      }

      // Zum nächsten Hook in der Hook-Kette wechseln
      return CallNextHookEx(hHook, code, wParam, lParam);
   }

   /* Zeigt die MessageBox an */
   public static ExtDialogResult Show(string text, string
      caption, string button1Caption, string button2Caption,
      string button3Caption, MessageBoxIcon icon,
      MessageBoxDefaultButton defaultButton)
   {
      // Festlegen der als Basis verwendeten Schalter
      MessageBoxButtons buttons;
      if (button3Caption != null)
      {
         buttons = MessageBoxButtons.YesNoCancel;
      }
      else if (button2Caption != null)
      {
         buttons = MessageBoxButtons.YesNo;
      }
      else
      {
         buttons = MessageBoxButtons.OK;
      }

      // Schalterbeschriftungen an die privaten Felder
      // übergeben, damit diese in der Hook-Funktion zur
      // Verfügung stehen
      ExtMessageBox.button1Caption = button1Caption;
      ExtMessageBox.button2Caption = button2Caption;
      ExtMessageBox.button3Caption = button3Caption;
```

Listing 500: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

```
      // CBT-Hook nur für den aktuellen Thread installieren
      ExtDialogResult returnValue;
      try
      {
         hHook = SetWindowsHookEx(HookType.WH_CBT,
            new HookProcDelegate(HookProc), IntPtr.Zero,
            (int)AppDomain.GetCurrentThreadId());

         // Standard-MessageBox aufrufen
         switch (MessageBox.Show(text, caption, buttons, icon,
            defaultButton))
         {
            case DialogResult.OK:
               returnValue = ExtDialogResult.FirstButton;
               break;
            case DialogResult.Yes:
               returnValue = ExtDialogResult.FirstButton;
               break;
            case DialogResult.No:
               returnValue = ExtDialogResult.SecondButton;
               break;
            case DialogResult.Cancel:
               returnValue = ExtDialogResult.ThirdButton;
               break;
            default:
               returnValue = ExtDialogResult.FirstButton;
               break;
         }
      }
      finally
      {
         // Hook wieder deinstallieren
         UnhookWindowsHookEx(hHook);
      }

      // Ergebnis zurückgeben
      return returnValue;
   }
}
```

Listing 500: Klasse für eine erweiterte MessageBox mit der Möglichkeit, die Beschriftungen der Schalter zu definieren (Forts.)

Die Verwendung dieser Klasse ist dann sehr einfach:

```
// Die erweiterte Messagebox öffnen
string filename = "C:\\Log.txt";
switch (ExtMessageBox.Show("Wollen Sie die Datei '" + filename +
   "' neu erzeugen oder an die Datei anfügen?",
   Application.ProductName, "Erzeugen", "Anfügen", "Abbrechen",
   MessageBoxIcon.Question, MessageBoxDefaultButton.Button1))
{
   case ExtDialogResult.FirstButton:
      Console.WriteLine("'Datei erzeugen' wurde gewählt");
      break;

   case ExtDialogResult.SecondButton:
      Console.WriteLine("'Datei anfügen' wurde gewählt");
      break;

   case ExtDialogResult.ThirdButton:
      Console.WriteLine("'Abbrechen' wurde gewählt");
      break;
}
```

Listing 501: Anwendung der Klasse für eine erweiterte MessageBox

WPF

Dieses Kapitel enthält spezielle Rezepte für die Anwendungsentwicklung mit WPF. Auf WPF-Grundlagen gehe ich (natürlich) nicht ein. Gerade bei WPF ist es aber sehr schwierig, die Grundlagen von dem speziellen Wissen zu unterscheiden, das in ein Codebook gehört. Ich gehe aber davon aus, dass Sie die WPF-Grundlagen kennen. Und dazu gehören auch komplexere Themen wie Stile, Trigger und Vorlagen. Falls Sie Nachholbedarf haben, lesen Sie das PDF-Dokument »WPF-Grundlagen«, das Sie auf der Buch-CD im Ordner *Zusatz-Artikel* finden.

Viele der hier beschriebenen WPF-Rezepte gleichen übrigens Rezepten, die ich bereits im `Windows.Forms`-Kapitel beschrieben habe. Das ist Absicht, da ich zum einen das WPF-Kapitel in etwa mit dem `Windows.Forms`-Kapitel synchronisieren und zum anderen WPF sauber von `Windows.Forms` trennen wollte. Dabei entstanden einige unter WPF eher triviale Rezepte wie »260 Fenster ohne Titelleiste« und »262 Fenster über den Clientbereich verschiebar machen«.

Einige `Windows.Forms`-Rezepte konnte bzw. musste ich auch nicht nach WPF umsetzen. In der folgenden Auflistung finden Sie die Gründe:

- »Andockende Formulare«: Für eine entsprechende Lösung unter WPF ist kein Platz im Buch. Eine hervorragende Bibliothek dazu finden Sie an der Adresse *www.codeproject.com/KB/WPF/WPFdockinglib.aspx*. Diese Bibliothek im Buch zu erläutern wäre zu aufwändig.
- »Ändern des Systemmenüs – Entfernen des Schließen-Eintrags und -Schalters«: Ehrlich gesagt zweifle ich mittlerweile an dem Sinn dieses Rezepts ☺.
- »Ermitteln, auf welchen Monitoren ein Formular angezeigt wird«: Ich bin mir nicht sicher, ob ein solches Rezept unter WPF Sinn machen würde.
- »Menüs in der Laufzeit erstellen oder erweitern«: Das grundsätzliche Vorgehen entspricht unter WPF dem in einer `Windows.Forms`-Anwendung.
- »Steuerelemente mit transparentem Hintergrund«: Hier gilt dasselbe wie beim vorherigen Punkt.
- »Die Position des Eingabecursors in einer TextBox, RichTextBox oder MaskedTextBox ermitteln«: Die Umsetzung ist nicht möglich, da WPF-Steuerelemente keinen Handle besitzen (weil sie von WPF gezeichnet werden).
- »Die angezeigten Zeilen einer MultiLine-TextBox auslesen«: Hier besteht dasselbe Problem wie zuvor.
- »ComboBox mit Autovervollständigung«: Dieses Rezept ist nicht notwendig, weil die Autovervollständigung der WPF-`ComboBox` (im Gegensatz zu der `Windows.Forms`-Version) hervorragend funktioniert, da diese die originale Liste für die Autovervollständigung verwendet.
- »Daten neben den Einträgen einer ListBox oder ComboBox verwalten«: Dieses Rezept ist zu trivial, da Inhalts- und Listensteuerelemente in WPF grundsätzlich beliebige Objekte als Inhalt besitzen können.
- »ListBox mit ToolTip für längere Einträge«: Ich denke nicht, dass ein ähnliches Rezept für WPF sinnvoll wäre, da eine `ListBox` (über ihre `ItemTemplate`-Eigenschaft) problemlos mit

einer beliebigen Vorlage für ihre Einträge versehen werden kann (deren Steuerelemente per Datenbindung an die aufgelisteten Objekte gebunden werden). So können Sie z.B. problemlos `TextBlock`-Elemente in der Liste anzeigen um Text automatisch umbrochen darzustellen.

- »ListView sortieren«: Dieses Rezept macht unter WPF in meinen Augen nicht viel Sinn, da ein `ListView` (wie alle Listen-Steuerelemente) idealerweise per Datenbindung an eine Datenquelle gebunden wird und Sie diese recht einfach sortieren können.
- »Knoten einer Ebene in einem TreeView-Steuerelement vertauschen«: Für dieses Rezept blieb kein Platz.
- »Einzelne Knoten eines TreeView-Steuerelements sortieren«: Hier gilt dasselbe wie zuvor.
- Alle Rezepte für das DataGridView: Diese Rezepte sind auf das `Windows.Forms-DataGridView` spezialisiert.
- »Feiertage im MonthCalendar-Steuerelement darstellen«: Ein solches Steuerelement gibt es (noch) nicht unter WPF.
- »Ein Ordner-Dialog«: WPF besitzt zurzeit selbst noch keinen Ordner-Dialog. Um unter WPF einen solchen zu verwenden, müssen Sie auf den `Windows.Forms`-Dialog zurückgreifen.
- »In einem eigenen Steuerelement verhindern, dass die Cursor-Tasten einen Fokuswechsel bewirken«: Dieses Rezept ist zu speziell auf `Windows.Forms` zugeschnitten.
- »Hooking – Eine MessageBox mit definierten Schalterbeschriftungen versehen«: Das ist ein sehr spezielles Rezept. Wenn Sie eine `MessageBox` mit definierten Schalterbeschriftungen benötigen, nehmen Sie einfach die im `Windows.Forms`-Rezept beschriebene.

260 Fenster ohne Titelleiste

In einer WPF-Anwendung erhalten Sie ein Fenster ohne Titelleiste, indem Sie die Eigenschaft `WindowStyle` auf `None` setzen.

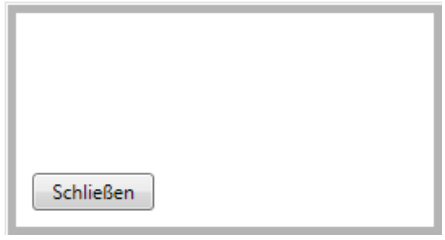


Abbildung 158: WPF-Fenster mit WindowStyle = None

261 Den Handle eines WPF-Fensters ermitteln

Für den Aufruf von API-Funktionen auf einem WPF-Fenster benötigen Sie dessen Fenster-Handle. Ein Beispiel dafür finden Sie schon im Rezept 262. Die Klasse `Window` stellt aber keine solche Eigenschaft zur Verfügung. Den Handle können Sie aber trotzdem über eine Instanz der Klasse `WindowInteropHelper` ermitteln:

```
System.Windows.Window window = ...;
System.IntPtr windowHandle =
   new System.Windows.Interop.WindowInteropHelper(window).Handle;
```

Listing 502: Ermitteln des Handle eines WPF-Fensters

Achtung Beachten Sie, dass Sie den Handle eines WPF-Fensters nicht im Konstruktor ermitteln können. Verwenden Sie dazu das `SourceInitialized`- oder `Loaded`-Ereignis.

262 Fenster über den Clientbereich verschiebbar machen

Ein Fenster mit `WindowStyle = None` (siehe Rezept 260) oder mit einer besonderen Form (siehe Rezept 267) kann vom Anwender nicht verschoben werden, da die Titelleiste fehlt. Dieses Problem können Sie aber lösen, indem Sie dem Anwender ermöglichen, das Formular über den Clientbereich zu verschieben. Diese (unter WPF sehr einfache) Technik können Sie natürlich auch bei normalen Fenstern einsetzen. Dazu fangen Sie linke Maustaste ab und rufen die `DragMove`-Methode des Fensters auf:

```
private void Window_MouseLeftButtonDown(object sender,
   System.Windows.Input.MouseButtonEventArgs e)
{
   // Das Verschieben des Fensters über die Maus starten
   this.DragMove();
}
```

Listing 503: Das Verschieben eines Fensters über die Maus bei der Betätigung der linken Maustaste starten

Tipp Wenn gleichzeitig andere Steuerelemente auf dem Fenster die linke Maustaste abfangen sollen, können Sie deren Ereignisse `PreviewMouseLeftButtonDown` und `PreviewMouseDown` verwenden, um die Weitergabe der Maus an das Fenster ggf. über `e.Handled = true` abbrechen zu können und damit das Verschieben zu vermeiden.

263 Windows-Nachrichten verarbeiten

In einem WPF-Fenster haben Sie nicht wie unter `Windows.Forms` die direkte Möglichkeit, Nachrichten zu verarbeiten, die Windows dem Fenster sendet. Für einige Tricks wie das Maximieren eines titellosen Fensters unter Berücksichtigung der Windows-Taskbar (Rezept 264) ist aber die Verarbeitung dieser Nachrichten notwendig.

Um Windows-Nachrichten unter WPF abfangen zu können, müssen Sie eine »Hook«-Methode (Hook = Haken) hinzufügen. Diese Methode hängt sich in die Nachrichtenverarbeitung ein und kann somit alle Windows-Nachrichten verarbeiten. Die Hook-Methode muss die folgende Signatur aufweisen:

```
private IntPtr WindowProc(IntPtr hwnd, int msg,
   IntPtr wParam, IntPtr lParam, ref bool handled)
```

Die Argumente sind die folgenden:

- *hwnd*: der Handle des Fensters
- *msg*: die Nachricht
- *wParam*: Zusatzinformationen, die von der Nachricht abhängen
- *lParam*: weitere, von der Nachricht abhängende Zusatzinformationen
- *handled*: Dieses Argument kann auf `true` gesetzt werden um die Nachrichtenverarbeitung zu beenden. Nachfolgende Hooks werden dann nicht mehr ausgeführt.

Die Rückgabe der Hook-Methode hängt von der jeweiligen Windows-Nachricht ab. Die Default-Rückgabe ist `IntPtr.Zero`.

Die Hook-Methode können Sie nicht direkt in ein WPF-Fenster einhängen. Der Trick ist aber, dazu eine `HwndSource`-Instanz zu verwenden, die ein Windows-Fenster repräsentiert. Das Windows-Fenster eines WPF-Fensters erhalten Sie über die statische `FromHwnd`-Methode dieser Klasse, der Sie den Handle des WPF-Fensters (siehe Rezept 261) übergeben.

Über die `AddHook`-Methode der `HwndSource`-Instanz hängen Sie dann den Hook ein. Dies programmieren Sie idealerweise im `SourceInitialized`-Ereignis des Fensters. Dieses Ereignis wird aufgerufen, nachdem das Windows-Fenster erzeugt wurde (und bevor es angezeigt wird).

Das folgende Listing zeigt eine Nachrichtenverarbeitung an einem funktionslosen Beispiel. In Rezept 264 finden Sie eine Anwendung dieser Technik.

Zum Kompilieren des Programmcodes müssen Sie die Namensräume `System`, `System.Windows` und `System.Windows.Interop` einbinden.

```
private void Window_SourceInitialized(object sender, EventArgs e)
{
   // Die Hook-Methode in die Nachrichtenverarbeitung einhängen
   HwndSource hwndSource = HwndSource.FromHwnd(
      new WindowInteropHelper(this).Handle);
   hwndSource.AddHook(new HwndSourceHook(this.WindowProc));
}

/* Der Hook-Handler für den Windows-Nachrichten-Hook */
private IntPtr WindowProc(IntPtr hwnd, int msg,
   IntPtr wParam, IntPtr lParam, ref bool handled)
{
   // Zunächst festlegen, dass die Nachricht nicht als verarbeitet gilt
   handled = false;
```

Listing 504: Einhängen einer Hook-Methode zur Verarbeitung der Nachrichten eines Fensters

```
   // Die Nachricht auswerten
   switch (msg)
   {
      case ...
        // Die Nachricht verarbeiten
        ...

        // Die weitere Verarbeitung abbrechen (hängt von der Nachricht ab)
        handled = true;

        // Einen Ergebniswert zurückgeben (hängt von der Nachricht ab)
        return ...
   }

   // Ein leeres Ergebnis zurückgeben
   return IntPtr.Zero;
}
```

Listing 504: Einhängen einer Hook-Methode zur Verarbeitung der Nachrichten eines Fensters (Forts.)

264 Beim Maximieren eines Fensters ohne Titelleiste die Taskbar berücksichtigen

Wenn Sie ein Fenster ohne Titelleiste (`WindowStyle = None`) maximieren, nimmt dieses den gesamten Platz des Bildschirms in Anspruch. Bei einem Fenster mit Titelleiste passiert das nicht. Wenn Sie nichts davon halten, die Windows Taskbar mit einem maximierten Fenster ohne Titelleiste zu überdecken, können Sie dieses Rezept einsetzen, um ein solches Fenster nur auf den verfügbaren Platz zu maximieren.

Eine denkbare Lösung des Problems wäre, das Fenster über seine `Width`- und `Height`-Eigenschaft auf den verfügbaren Platz zu maximieren (anstatt `WindowState` auf `WindowState.Maximized` zu setzen). Dabei haben Sie aber das Problem, zunächst den Bildschirm zu ermitteln, auf dem das Fenster liegt, und dann noch dessen Arbeitsbereich. Das ist alles möglich, die in diesem Rezept verwendete Lösung ist aber allgemeiner, weil sie das Maximieren eines Fensters abfängt und die Größe neu definiert. Und diese Lösung funktioniert auch in einem Mehr-Monitor-System, wenn das Fenster auf einem anderen als dem primären Bildschirm geöffnet wird.

Die in diesem Rezept beschriebene Lösung des Problems ist, die Windows-Nachricht `WM_GETMINMAXINFO` (0x0024) abzufangen, die dem Fenster gesendet wird, wenn es maximiert wird. Am Argument `lParam` dieser Nachricht übergibt Windows einen Zeiger auf eine Instanz der Struktur `MINMAXINFO`, die die Default-Position und -Größe beinhaltet. Das Fenster kann diese überschreiben, indem es einen Zeiger auf eine neue oder veränderte `MINMAXINFO`-Instanz in das Argument `lParam` zurückschreibt.

Für die Programmierung ist nun noch das Ermitteln des Bildschirms notwendig, auf dem das Fenster liegt. Dazu verwendet dieses Rezept die API-Funktion `MonitorFromWindow`, die in der Lage ist, einen Zeiger auf eine Monitor-Information für den Monitor zurückzugeben, der den größten Teil des Fensters enthält. Dazu werden dieser Methode das Handle des Fensters und die Konstante `MONITOR_DEFAULTTONEAREST` (0x0002) übergeben. Der zurückgegebene `IntPtr`-Zeigerwert wird dann über die `GetMonitorInfo`-Funktion in eine `MONITORINFO`-Instanz geschrieben. Über diese ermittelt Listing 505 den Arbeitsbereich des Monitors, erzeugt damit eine neue `MINMAXINFO`-Instanz und schreibt diese schließlich als Zeiger in `lParam` zurück.

Zum Kompilieren des Programmcodes benötigen Sie (natürlich) ein Fenster (☺) und den Import der Namensräume `System`, `System.Runtime.InteropServices`, `System.Windows` und `System.Windows.Interop`.

Zunächst sind einige API-Deklarationen notwendig:

```
[StructLayout(LayoutKind.Sequential)]
public struct POINT
{
   public int x;
   public int y;
}

[StructLayout(LayoutKind.Sequential)]
public struct MINMAXINFO
{
   public POINT ptReserved;
   public POINT ptMaxSize;
   public POINT ptMaxPosition;
   public POINT ptMinTrackSize;
   public POINT ptMaxTrackSize;
};

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
public class MONITORINFO
{
   public int cbSize = Marshal.SizeOf(typeof(MONITORINFO));
   public RECT rcMonitor = new RECT();
   public RECT rcWork = new RECT();
   public int dwFlags = 0;
}

[StructLayout(LayoutKind.Sequential, Pack = 0)]
public struct RECT
{
```

Listing 505: Deklaration der notwendigen API-Funktionen und -Strukturen für das Maximieren

```
   public int left;
   public int top;
   public int right;
   public int bottom;
}

[DllImport("user32")]
internal static extern bool GetMonitorInfo(IntPtr hMonitor, MONITORINFO lpmi);

[DllImport("User32")]
internal static extern IntPtr MonitorFromWindow(IntPtr handle, int flags);
```

Listing 505: Deklaration der notwendigen API-Funktionen und -Strukturen für das Maximieren (Forts.)

Im `SourceInitialized`-Ereignis des Fensters hängen Sie die Nachrichten-Verarbeitungs-Hook-Methode ein:

```
private void Window_SourceInitialized(object sender, EventArgs e)
{
   WindowInteropHelper helper = new WindowInteropHelper(this);
   HwndSource hwndSource = HwndSource.FromHwnd(helper.Handle);
   hwndSource.AddHook(new HwndSourceHook(this.WindowsProc));
}
```

Listing 506: Einhängen des Hook-Handlers für die Nachrichtenverarbeitung

In der Hook-Methode schließlich fangen Sie die `WM_GETMINMAXINFO`-Nachricht ab und verändern ggf. die übergebene `MINMAXINFO`-Struktur:

```
private System.IntPtr WindowsProc(System.IntPtr hwnd,
   int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
{
   const int WM_GETMINMAXINFO = 0x0024;

   switch (msg)
   {
      case WM_GETMINMAXINFO:
         // Die am Argument lParam als Zeiger übergebene MINMAXINFO-Struktur
         // auslesen
         MINMAXINFO minMaxInfo = (MINMAXINFO)Marshal.PtrToStructure(lParam,
            typeof(MINMAXINFO));
```

Listing 507: Abfangen der WM_GETMINMAXINFO-Nachricht, um die Maximalgröße eines Fensters auf den Arbeitsbereich einzuschränken

```
            // Einen Handle auf den Monitor holen, der den größten Teil
            // des Fensters besitzt
            int MONITOR_DEFAULTTONEAREST = 0x0002;
            IntPtr monitor = MonitorFromWindow(hwnd, MONITOR_DEFAULTTONEAREST);

            if (monitor != System.IntPtr.Zero)
            {
               // Informationen zu diesem Monitor einlesen
               MONITORINFO monitorInfo = new MONITORINFO();
               GetMonitorInfo(monitor, monitorInfo);
               RECT workArea = monitorInfo.rcWork;
               RECT monitorArea = monitorInfo.rcMonitor;

               // Die MINMAXINFO-Struktur mit der Position und Größe
               // des Arbeitsbereichs des Bildschirms belegen
               minMaxInfo.ptMaxPosition.x = Math.Abs(
                  workArea.left - monitorArea.left);
               minMaxInfo.ptMaxPosition.y = Math.Abs(
                  workArea.top - monitorArea.top);
               minMaxInfo.ptMaxSize.x = Math.Abs(workArea.right - workArea.left);
               minMaxInfo.ptMaxSize.y = Math.Abs(workArea.bottom - workArea.top);
            }

            // Einen Zeiger auf die MinMaxInfo-Struktur in lParam schreiben
            Marshal.StructureToPtr(minMaxInfo, lParam, true);

            // Die Nachricht als behandelt kennzeichnen
            handled = true;

            break;
      }

      // Den Defaultwert für Nachrichten zurückgeben
      return IntPtr.Zero;
}
```

Listing 507: Abfangen der WM_GETMINMAXINFO-Nachricht, um die Maximalgröße eines Fensters auf den Arbeitsbereich einzuschränken (Forts.)

Hinweis

Wenn Sie ein titelloses Fenster mit dem Programmcode aus diesem Rezept ausstatten und direkt beim Öffnen maximiert darstellen wollen, können Sie dazu nicht im XAML-Code die Eigenschaft `WindowState` mit `Maximized` vorbelegen. In diesem Fall würde das Fenster trotz der Programmierung auf den gesamten Bereich des Bildschirms vergrößert. Setzen Sie diese Eigenschaft im `Loaded`-Ereignis des Fensters.

265 Fenster verlaufend füllen

Unter WPF ist das Füllen eines Fensters mit einer verlaufenden Hintergrundfarbe eine eher triviale Angelegenheit (anders als unter `Windows.Forms`). Dazu schreiben Sie einen `LinearGradientBrush` oder `RadialGradientBrush` in die `Background`-Eigenschaft des Fensters. Das einzig Schwierige daran ist die Definition des Pinsels. Ich gehe hier auf einen linearen Verlauf mit einem `LinearGradientBrush` ein.

Ein `LinearGradientBrush` besitzt einen Start- und einen Endpunkt. Diese werden mit logischen Koordinaten im Bereich von 0 bis 1 angegeben. 0 steht dabei für links bzw. oben, 1 für rechts bzw. unten. 0,5 ist demnach auf der X- und Y-Achse die Mitte.

Ein linearer Verlauf besitzt zumindest zwei Stopps, die über `GradiantStop`-Instanzen definiert werden. Die `Offset`-Eigenschaft eines `GradiantStop`-Objekts gibt an, an welcher Position dieser Stopp liegt. Der Offset bezieht sich auf den linearen Verlauf. 0 steht für den Anfang, 1 für das Ende. Alle Werte dazwischen stehen für relative Positionen innerhalb des Verlaufs. Um einen Verlauf mit mehreren Farben oder geteilte Verläufe zu erzeugen, können Sie auch mehr als zwei `GradiantStop`-Objekte angeben.

Das folgende Beispiel erzeugt einen einfachen Verlauf von links nach rechts:

```
<Window x:Class="Fenster_verlaufend_füllen.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Fenster verlaufend füllen" Height="179" Width="427">
   <Window.Background>
      <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">
         <GradientStop Color="Blue" Offset="0" />
         <GradientStop Color="#000040" Offset="1" />
      </LinearGradientBrush>
   </Window.Background>
   <Grid>

    </Grid>
</Window>
```

Listing 508: Ein einfacher Verlauf auf einem WPF-Fenster

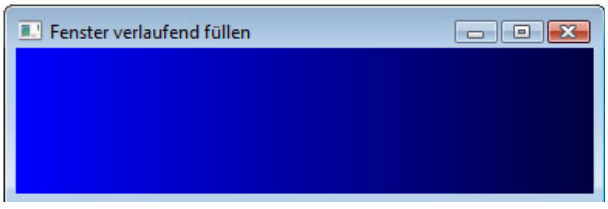


Abbildung 159: Das Verlaufs-Fenster in Aktion

266 Hintergrund mit Textur

Wenn Sie ein Fenster oder Steuerelement mit einem Textur-Hintergrund erstellen wollen, ist das Vorgehen und WPF zwar sehr flexibel, aber nicht so einfach wie unter `Windows.Forms`. Deshalb ist dieses Thema ein Rezept wert.

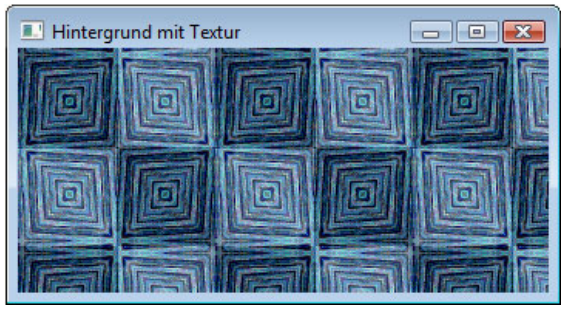


Abbildung 160: Beispiel für einen Hintergrund mit einer Textur

Unter WPF erzeugen Sie Texturen mit einer von `TileBrush` abgeleiteten Klasse. In der aktuellen Version sind das die Klassen `DrawingBrush`, `ImageBrush` und `VisualBrush`. Über einen `ImageBrush` können Sie zum Beispiel ein Bild gekachelt darstellen.

Die `TileMode`-Eigenschaft bestimmt, wie die Grafik des Pinsels gekachelt wird:

- `TileMode.None`: Die Grafik wird nicht gekachelt, was eigentlich nur heißt, dass es genau eine Kachel gibt
- `TileMode.Tile`: Die Grafik wird gekachelt, indem die einzelnen Grafik nebeneinander gestellt werden
- `TileMode.FlipX`: Die Grafik wird gekachelt. Die Kacheln werden abwechselnd um die X-Achse gedreht
- `TileMode.FlipY`: Die Grafik wird gekachelt, wobei die Kacheln abwechselnd um die Y-Achse gedreht werden
- `TileMode.FlipXY`: die Grafik wird gekachelt. Die Kacheln werden abwechselnd um die X- und die Y-Achse gedreht.

Die Eigenschaft `Stretch` bestimmt, wie die Grafik des Pinsels gedehnt wird:

- `Stretch.None`: Die Grafik wird nicht gedehnt
- `Stretch.Fill`: Die Grafik wird so gedehnt, dass sie den kompletten Innenraum der Kachel umfasst
- `Stretch.Uniform`: Die Grafik wird so gedehnt, dass sie möglichst den kompletten Innenraum der Kachel umfasst, dabei werden aber die Proportionen beibehalten
- `Stretch.UniformToFill`: Ähnlich wie `Uniform`, nur dass die Grafik zudem so vergrößert wird, dass sie den kompletten Innenraum der Kachel umfasst

Alleine das Setzen von TileMode und Stretch bringt für einen Textur-Hintergrund noch nichts. Wenn Sie TileMode z. B. auf Tile und Stretch auf None setzen, wird die Grafik des Pinsels nur ein einziges Mal angezeigt.

Die Ursache dafür ist der ViewPort des Pinsels. Der ViewPort eines TileBrush bestimmt die Position und die Ausmaße einer Basis-Kachel. Per Voreinstellung stehen dessen Eigenschaften Left und Top auf 0 und Right und Bottom auf 1, was in der Voreinstellung bedeutet, dass dieser den kompletten Innenraum des Fensters bzw. Steuerelements umfasst. Eine Kachel ist per Voreinstellung also immer genauso groß wie das Steuerelement bzw. Fenster.

Den ViewPort – und damit die Kachel-Grundposition und -Größe – können Sie natürlich einstellen. Dabei werden zwei Modi unterschieden, die in der Eigenschaft ViewPortUnits eingestellt werden:

- BrushMappingMode.RelativeToBoundingBox: Steht dafür, dass die ViewPort-Angaben relativ sind. Der Wert 0 repräsentiert dann links bzw. oben, 1 steht für rechts bzw. unten. Dazwischen sind natürlich auch alle Zwischenwerte möglich. Dies ist die Voreinstellung.
- BrushMappingMode.Absolute: Gibt an, dass die Angaben mit absoluten Werten erfolgen.

Um ein Bild gekachelt auszugeben, können Sie nun mehrere Wege gehen: Sie können den ViewPort auf eine beliebige Größe einstellen und mit Stretch = Stretch.UniformToFill erreichen, dass das Bild auf die Kachelgröße angepasst wird, ohne die Proportionen zu verändern. Besser ist aber, die Kachelgröße so einzustellen, dass diese genau der Größe des Bilds entspricht. Dazu setzen Sie ViewPortUnits auf BrushMappingMode.Absolute und stellen den ViewPort entsprechend der Größe des Bilds ein. Dabei können Sie die Größe durch einfaches Ausprobieren ermitteln und mit einem festen Wert zuweisen:

```
<Grid>
   <Grid.Background>
      <ImageBrush ImageSource="Texture.jpg"
         Stretch="None" TileMode="Tile"
         Viewport="0,0,125,125" ViewportUnits="Absolute" />
   </Grid.Background>
</Grid>
```

Listing 509: Grid mit einem gekachelten Hintergrund

Eine in meinen Augen bessere Möglichkeit ist, die tatsächliche Größe des Bilds auszulesen und diese als Kachelgröße zu verwenden. In XAML können Sie dies aber nicht direkt (über Datenbindung) umsetzen, da die für ViewPort verwendete Rect-Struktur einfache Eigenschaften einsetzt (keine Abhängigkeitseigenschaften), an die keine Datenbindung möglich ist.

Sie können aber einen eigenen Konverter einsetzen, der einen ImageBrush in eine Rect-Struktur »konvertiert«, und diesen in der Datenbindung verwenden. Den Konverter legen Sie in einer separaten Klasse an:

```
using System;
using System.Windows;
using System.Windows.Data;
using System.Windows.Media;

public class ImageBrushRectConverter : IValueConverter
{
   public object Convert(object value, Type targetType,
      object parameter, System.Globalization.CultureInfo culture)
   {
      ImageBrush imageBrush = (ImageBrush)value;
      ImageSource source = imageBrush.ImageSource;
      return new Rect(0, 0, source.Width, source.Height);
   }

   public object ConvertBack(object value, Type targetType, object parameter,
      System.Globalization.CultureInfo culture)
   {
      throw new NotImplementedException();
   }
}
```

Listing 510: Konverter zum Konvertieren eines ImageBrush-Objekts in eine Rect-Instanz mit den Ausmaßen des Bilds des ImageBrush

In XAML verwenden Sie dann Datenbindung, um die `ViewPort`-Eigenschaft an das über den Konverter umgesetzte `ImageBrush`-Objekt zu binden:

```
<Window x:Class="Hintergrund_mit_Textur.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Hintergrund_mit_Textur"
    Title="Hintergrund mit Textur" Height="300" Width="500" >
   <Window.Resources>
      <local:ImageBrushRectConverter x:Key="imageBrushRectConverter"/>
   </Window.Resources>
    <Grid>
      <Grid.Background>
         <ImageBrush ImageSource="Texture.gif"
            Stretch="None" TileMode="Tile"
            Viewport="{Binding RelativeSource={RelativeSource Self},
               Converter={StaticResource imageBrushRectConverter}}"
            ViewportUnits="Absolute" />
```

Listing 511: Gekachelter Hintergrund mit automatischer ViewPort-Einstellung über den Konverter

```
      </Grid.Background>
   </Grid>
</Window>
```

Listing 511: Gekachelter Hintergrund mit automatischer ViewPort-Einstellung über den Konverter (Forts.)

Eine andere Lösung wäre die Definition des `ViewPort` im Programm. In diesem Fall können Sie auch gleich den kompletten Hintergrund im Programm definieren:

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;

...

// Das Bild aus der Ressource laden
BitmapImage bitmapImage = new BitmapImage(
  new Uri("pack://application:,,,/Texture.gif"));

// ImageBrush erstellen und einstellen
ImageBrush imageBrush = new ImageBrush(bitmapImage);
imageBrush.TileMode = TileMode.Tile;
imageBrush.ViewportUnits = BrushMappingMode.Absolute;
imageBrush.Viewport = new Rect(0, 0, bitmapImage.Width,
   bitmapImage.Height);

// Den ImageBrush als Hintergrund setzen
this.Background = imageBrush;
```

Listing 512: Definition eines gekachelten Hintergrunds im Programm

Hinweis

Falls Sie den URI nicht verstehen, der `BitmapImage` übergeben wird, lesen Sie im Artikel »WPF-Grundlagen« nach, den Sie auf der Buch-CD finden. Suchen Sie im Index nach »Paket-URI« um eine Erläuterung zu finden.

267 Fenster mit speziellen Formen

Fenster mit einer nicht rechteckigen Form können Sie unter WPF sehr einfach erstellen, indem Sie den Hintergrund des Fensters transparent schalten und auf dem Fenster beliebige Objekte anlegen. Dazu müssen Sie die folgenden Eigenschaften des Fensters einstellen:

- `WindowStyle`: `None`
- `AllowsTransparency`: `True`
- `Background`: `Transparent`

Dann legen Sie einfach verschiedene Elemente auf dem Fenster ab. Da der transparente Teil eines Fensters für die Maus nicht berücksichtigt wird, ist dieser für den Benutzer auch beim Klicken mit der Maus nicht vorhanden.

Das folgende Beispiel definiert das Fenster, das in Abbildung 161 dargestellt wird.

```
<Window x:Class="Fenster_mit_speziellen_Formen.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="250" Width="422"
    WindowStyle="None"
    AllowsTransparency="True"
    Background="Transparent" >
   <Canvas>
      <Rectangle Canvas.Left="0" Canvas.Top="45" Width="400" Height="120"
         Fill="Orange" Stroke="Black" StrokeThickness="2" />
      <Ellipse Canvas.Left="100" Canvas.Top="0" Width="200" Height="200"
         Fill="Navy" Stroke="Black" StrokeThickness="2" />

      <Button Name="btnClose" Width="100" Height="23"
         Canvas.Left="150" Canvas.Top="129"
         Click="btnClose_Click">Schließen</Button>
   </Canvas>
</Window>
```

Listing 513: Fenster mit transparentem Hintergrund und einigen WPF-Elementen

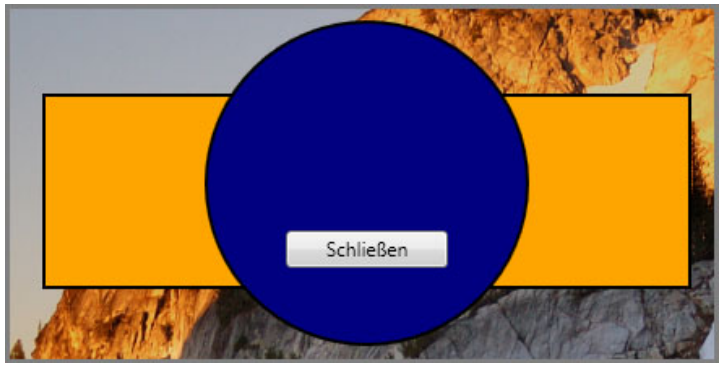


Abbildung 161: Das Beispielfenster vor einem der Vista-Standard-Hintergründe

Tipp Nutzen Sie das Rezept 262, um das Fenster über seinen (nicht transparenten) Clientbereich verschiebbar zu machen.

Hinweis

In dem Beispiel zu diesem Rezept finden Sie auch ein Fenster mit einem schöneren Hintergrund (mit Bildern).

268 Fenster mit dem Vista-Glas-Effekt ausstatten

WPF-Fenster unterstützen den Vista-Glas-Effekt (im Aero-Thema) leider nicht direkt für ein komplettes Fenster. Dabei wäre es in einigen Fällen wünschenswert, ein Fenster wie in Abbildung 162 komplett »verglast« darzustellen.



Abbildung 162: Ein vollverglastes Fenster vor einem der Vista-Standard-Hintergründe

Um dies zu erreichen, müssen Sie zwei API-Funktionen einsetzen: Die Funktion `DwmIsCompositionEnabled` überprüft, ob die »Desktopgestaltung« (Desktop composition) von Vista zurzeit eingeschaltet ist, die u. a. für den Glas-Effekt zuständig ist. Die Desktopgestaltung ist im Vista-Basis-Thema nicht eingeschaltet und kann auch separat abgeschaltet werden.

Die Funktion `DwmExtendFrameIntoClientArea` erweitert den Glas-Rahmen des Fensters in den Clientbereich hinein. Am ersten Argument übergeben Sie dieser Funktion den Handle des Fensters (den Sie über das Rezept 261 ermitteln können). Am zweiten übergeben Sie eine `MARGINS`-Struktur, die festlegt, um wie viele Pixel der Rahmen nach innen an jeder Seite in den Innenbereich vergrößert werden soll. Damit das Ganze funktioniert, muss das Fenster transparent sein. Und natürlich muss das Aero-Thema aktiviert sein. Bei einem nicht aktivierten Aero-Thema wird allerdings der solide Rahmen nach innen vergrößert, sodass keine Fehler entstehen.

Die Methode `ExtendWindowFrame` in Listing 514 setzt dies um. Diese Methode erweitert den (ggf. im Glas-Design erscheinenden) Rahmen eines Fensters nach innen. Sie können der zweiten Variante am Argument *margins* ein `Thickness`-Objekt übergeben, das die Größe der Verbreiterung bestimmt. Damit können Sie auch Fenster erzeugen, deren Innenbereich transparent ist. Eine `Thickness`-Instanz, deren Werte mit -1 initialisiert sind, führt zu einer »Vollverglasung« des Fensters.

`ExtendWindowFrame` überprüft zunächst, ob die notwendige Desktopgestaltung aktiviert ist. Dabei wird gleich die `DllNotFoundException` abgefangen, die geworfen wird, wenn die DLL *dwmapi.dll*, die die genannten API-Funktionen enthält, nicht gefunden wird (weil die Anwendung auf einer älteren Windows-Version ausgeführt wird).

Dann werden der Handle des Fensters ermittelt und der Hintergrund des Fensters transparent geschaltet. Meine Version dieser im Internet in verschiedenen Blogs kursierenden Technik berechnet noch den Faktor für die Umrechnung der geräteunabhängigen Einheit von WPF in Pixel, um diesen Faktor bei der Berechnung der Pixeldaten für die `MARGINS`-Struktur verwenden zu können (nicht, dass ich darauf stolz bin, aber selbst Adam Nathan, der Auto von »Windows Presentation Foundation Unleashed« hat in seinem Blog-Beitrag nicht daran gedacht, Bildschirme mit anderen DPI-Werten als 96 zu berücksichtigen ☺).

Schließlich wird die `MARGINS`-Struktur zusammengestellt und der `DwmExtendFrameIntoClientArea`-Methode übergeben.

Zum Kompilieren des Programmcodes müssen Sie die Namensräume `System`, `System.Runtime.InteropServices`, `System.Windows`, `System.Windows.Interop` und `System.Windows.Media` einbinden.


```
struct MARGINS
{
   public int Left;
   public int Right;
   public int Top;
   public int Bottom;
}

[DllImport("dwmapi.dll", PreserveSig = false)]
static extern void DwmExtendFrameIntoClientArea(IntPtr hwnd, ref MARGINS margins);

[DllImport("dwmapi.dll", PreserveSig = false)]
static extern bool DwmIsCompositionEnabled();

/* Erweitert den Rahmen eines Fensters unter Vista nach innen */
public static bool ExtendWindowFrame(Window window)
{
   return ExtendWindowFrame(window, new Thickness(-1));
}

/* Erweitert den Rahmen eines Fensters unter Vista nach innen */
public static bool ExtendWindowFrame(Window window, Thickness margins)
{
   try
   {
```

Listing 514: Methoden zum Erweitern des Vista-Rahmens in den Clientbereich eines Fensters

```
      // Überprüfen, ob die notwendige Desktopgestaltung aktiviert ist
      if (DwmIsCompositionEnabled() == false)
      {
         return false;
      }
   }
   catch (DllNotFoundException)
   {
      // Die DLL dwmapi.dll ist nicht verfügbar.
      // Wahrscheinlich läuft die Anwendung unter
      // einer älteren Windows-Version
      return false;
   }

   // Den Handle des Fensters ermitteln
   IntPtr hwnd = new WindowInteropHelper(window).Handle;
   if (hwnd == IntPtr.Zero)
   {
      throw new InvalidOperationException(
         "Das Fenster muss bereits angezeigt werden, " +
         "damit der Glas-Rahmen nach innen verbreitert " +
         "werden kann.");
   }

   // Den Hintergrund (für WPF und Windows) transparent schalten
   window.Background = Brushes.Transparent;
   HwndSource.FromHwnd(hwnd).CompositionTarget.BackgroundColor =
      Colors.Transparent;

   // Den Faktor für die Umrechnung der geräteunabhängigen Einheit in
   // Pixel ermitteln
   double pixelCalculationFactorX = 1;
   double pixelCalculationFactorY = 1;
   PresentationSource source = PresentationSource.FromVisual(window);
   if (source != null)
   {
      pixelCalculationFactorX =
         source.CompositionTarget.TransformToDevice.M11;
      pixelCalculationFactorY =
         source.CompositionTarget.TransformToDevice.M22;
   }

   // Den Glas-Rahmen nach innen verbreitern
   MARGINS apiMargins = new MARGINS();
   apiMargins.Left = (int)(margins.Left * pixelCalculationFactorX);
```

Listing 514: Methoden zum Erweitern des Vista-Rahmens in den Clientbereich eines Fensters (Forts.)

```
   apiMargins.Top = (int)(margins.Top * pixelCalculationFactorY);
   apiMargins.Right = (int)(margins.Right * pixelCalculationFactorX);
   apiMargins.Bottom = (int)(margins.Bottom * pixelCalculationFactorX);
   DwmExtendFrameIntoClientArea(hwnd, ref apiMargins);

   return true;
}
```

Listing 514: Methoden zum Erweitern des Vista-Rahmens in den Clientbereich eines Fensters (Forts.)

Was Sie bei der Anwendung der `ExtendGlassFrame`-Methoden beachten sollten, ist, dass Sie diese im `SourceInitialized`-Ereignis aufrufen sollten. Außerdem sollten Sie die Windows-Nachricht `DWMCOMPOSITIONCHANGED` abfangen um auf Wechsel des Windows-Themas während der Laufzeit Ihrer Anwendung zu reagieren:

```
/* Stattet das Fenster mit einem erweiterten Aero-Rahmen aus */
protected override void OnSourceInitialized(EventArgs e)
{
   base.OnSourceInitialized(e);

   // Den Vista-Rahmen in den Innenbereich verbreitern
   WindowUtils.ExtendWindowFrame(this);

   // Die Hook-Methode in die Nachrichtenverarbeitung einhängen
   HwndSource hwndSource = HwndSource.FromHwnd(
      new WindowInteropHelper(this).Handle);
   hwndSource.AddHook(new HwndSourceHook(this.WindowProc));
}

/* Fängt den Wechsel des Windows-Themas ab um den Rahmen
   neu zu definieren */
private IntPtr WindowProc(IntPtr hwnd, int msg,
   IntPtr wParam, IntPtr lParam, ref bool handled)
{
   const int DWMCOMPOSITIONCHANGED = 0x031E;
   if (msg == DWMCOMPOSITIONCHANGED)
   {
      // Den Rahmen neu definieren
      WindowUtils.ExtendWindowFrame(this);
   }

   // Ein leeres Ergebnis zurückgeben
   return IntPtr.Zero;
}
```

Listing 515: Erweitern des Vista-Rahmens in einem Fenster

Hinweis

Ein Problem, das in diesem Rezept nicht gelöst ist, ist, dass der Wechsel vom Aero- in das Basis-Thema nicht korrekt abgefangen wird: Der Hintergrund des Fensters wird in diesem Fall schwarz dargestellt.

269 Fenster in einer Schleife aktualisieren

Wenn Sie in einem Fenster (oder auf einer Seite) eine Schleife ausführen und in dieser Steuerelemente aktualisieren, die auf dem Fenster liegen, werden die Steuerelemente erst dann tatsächlich aktualisiert, wenn die Schleife beendet ist. Dieses Problem liegt daran, dass Windows dem Aktualisieren von Steuerelementen eine wesentlich niedrigere Priorität zuweist als ein gerade laufendes Programm. Die Nachrichten, dass das Fenster aktualisiert werden muss, landen zwar in der Nachrichten-Warteschlange des Fensters. Das Fenster ignoriert die Nachrichten aber so lange, bis das aktuelle Programm abgearbeitet ist.

Dieses Problem können Sie über zwei Techniken lösen: Entweder rufen Sie so etwas wie die `DoEvents`-Methode der `Application`-Klasse von `Windows.Forms` auf oder Sie programmieren einen Thread (bzw. rufen eine Methode asynchron auf).

DoEvents

Die Methode `DoEvents` der `Application`-Klasse von `Windows.Forms` macht nichts anderes, als die Nachrichtenschleife eines Fensters abzuarbeiten. Sie ruft dazu die Methode `RunMessageLoop` des aktuellen `ThreadContext` auf. Die Klasse `ThreadContext` ist leider intern und kann also von außen nicht verwendet werden.

Als eine Lösung des Problems könnten Sie die Assembly *Windows.Forms.dll* referenzieren und `Application.DoEvents` aufrufen. Sie können aber auch den folgenden Trick verwenden:

```
if (Application.Current != null)
{
   Application.Current.Dispatcher.Invoke(DispatcherPriority.Background,
      new Action(delegate { }));
}
```

Listing 516: Simulation von DoEvents

`Dispatcher.Invoke` ruft die am zweiten Argument übergebene Methode synchron auf (auch wenn diese Methode leer ist). Die am ersten Argument übergebene Priorität `DispatcherPriority.Background` sorgt dafür, dass die Methode erst dann ausgeführt wird, wenn alle im Leerlauf befindlichen Vorgänge abgeschlossen sind. Dieser kleine Trick sorgt also dafür, dass `Invoke` so lange wartet, bis die Nachrichtenwarteschlange des Fensters leer ist. Da die darin enthaltenen `WM_PAINT`-Nachrichten dann auch abgearbeitet wurden, werden alle Veränderungen der Oberfläche gezeichnet.

Die Abfrage auf `Application.Current != null` ist übrigens sehr wichtig. Damit verhindern Sie, dass der Aufruf der `Invoke`-Methode zu einer `NullReferenceException` führt, wenn die Anwendung während der Ausführung des Code heruntergefahren wird.

Achtung

Wenn Sie mit diesem Trick arbeiten, müssen Sie aufpassen: Da das Fenster dann auch auf Eingaben reagiert, kann es vorkommen, dass der Benutzer den Programmcode noch einmal ausführt. Der erste Ablauf wird dann so lange unterbrochen, bis der zweite beendet ist. Das kann in der Praxis natürlich erhebliche Probleme verursachen. Deshalb sollten Sie verhindern, dass der Programmcode erneut ausgeführt werden kann, während er gerade ausgeführt wird, z.B. indem Sie das Steuerelement deaktivieren, über das er gestartet wird.

Eine Aktualisierung eines Fensters in einer Schleife, die von einem `Button` (mit Namen *updateWithoutThreadButton*) aus gestartet wird, sieht dann z.B. so aus:

```
private void updateWithoutThreadButton_Click(object sender, RoutedEventArgs e)
{
   try
   {
      // Den Schalter deaktivieren
      ((Button)sender).IsEnabled = false;

      for (int i = 0; i <= 100; i++)
      {
         // Das Label aktualisieren
         this.infoLabel.Content = i.ToString();

         // Die ProgressBar aktualisieren
         this.progressBar.Value = i;

         // DoEvents über den synchronen Aufruf einer
         // leeren Methode simulieren
         if (Application.Current != null)
         {
            Application.Current.Dispatcher.Invoke(
              DispatcherPriority.Background,
              new Action(delegate { }));
         }

         // Kleine Pause zur Demo
         Thread.Sleep(10);
      }
   }
   finally
   {
      // Den Schalter aktivieren
      ((Button)sender).IsEnabled = true;
   }
}
```

Listing 517: Aktualisieren eines Fensters in einer Schleife

Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading`, `System.Windows`, `System.Windows.Controls` und `System.Windows.Threading`.

Die Thread-Lösung

Wesentlich eleganter als die `DoEvents`-Simulation ist, dass Sie die Schleife in einem Thread oder über eine asynchron ausgeführte Methode ausführen (die ja nichts anderes ist, als ein Thread). Dabei müssen Sie natürlich alle Regeln des Threading berücksichtigen, z.B. dass Sie Threads miteinander synchronisieren oder dass nur der Thread, der ein Steuerelement oder Fenster erzeugt hat, auf dieses zugreifen darf. Eine asynchron aufgerufene Methode hat gegenüber einem Thread den Vorteil, dass Sie das Ende des Aufrufs abfangen können. So können Sie das Steuerelement, das die Aktion startet, zu Anfang deaktivieren und am Ende wieder aktivieren, um eine mehrfache Ausführung zu verhindern.

Das folgende Beispiel enthält eine einfache Schleife, die nichts weiter macht als ein Label und eine `ProgressBar` zu aktualisieren. Die Aktualisierung wird über eine asynchron ausgeführte Methode ausgeführt. Um die goldene Windows-Regel einzuhalten, dass nur der Thread, der ein Steuerelement erzeugt hat, auf dieses zugreifen darf, erfolgt der Zugriff auf die Steuerelemente über deren `Dispatcher`.

Für das Beispiel benötigen Sie ein Fenster mit einem Label (*infoLabel*), einer `ProgressBar` (*progressBar*) und einem Schalter (*updateWithThreadButton*). Es erfordert außerdem den Import der Namensräume `System`, `System.Threading`, `System.Windows`, `System.Windows.Controls` und `System.Windows.Threading`.


```
private void updateWithThreadButton_Click(object sender, RoutedEventArgs e)
{
   // Den Schalter deaktivieren
   ((Button)sender).IsEnabled = false;

   // Delegat für die asynchrone Ausführung
   Action worker = () =>
   {
      for (int i = 0; i <= 100; i++)
      {
         // Das Label threadsicher aktualisieren
         this.infoLabel.Dispatcher.Invoke(DispatcherPriority.Normal,
            new Action(() =>
            {
               this.infoLabel.Content = i.ToString();
            }));

         // Die ProgressBar threadsicher aktualisieren
         this.progressBar.Dispatcher.Invoke(DispatcherPriority.Normal,
            new Action(() =>
            {
```

Listing 518: Ausführung einer Schleife mit Aktualisierung von Steuerelementen in einer asynchron ausgeführten Methode

```
            this.progressBar.Value = i;
          }));

        // Kleine Pause zur Demo
        Thread.Sleep(10);
      }
   };

   // Delegat für das Callback-Ereignis
   AsyncCallback callback = (result) =>
   {
      // Den Schalter threadsicher aktivieren
      ((Button)sender).Dispatcher.Invoke(DispatcherPriority.Normal,
          new Action(() =>
          {
             ((Button)sender).IsEnabled = true;
          }));
   };

   // Die Methode asynchron aufrufen
   worker.BeginInvoke(callback, null);
}}
```

Listing 518: Ausführung einer Schleife mit Aktualisierung von Steuerelementen in einer asynchron ausgeführten Methode (Forts.)

Dieser Code sieht aufgrund der verwendeten Lambda-Ausdrücke etwas »wild« aus. Ich denke, eine kleine Erläuterung ist hier sinnvoll:

Der zu Anfang erzeugte Delegat für die asynchrone Ausführung wird am Ende verwendet, um die Methode asynchron auszuführen. Diesem Delegaten weise ich einen Lambda-Ausdruck zu, der dem `Action`-Delegaten entspricht (der also keine Argumente und keine Rückgabe besitzt). In der anonymen Methode des Lambda-Ausdrucks erfolgt die eigentliche Arbeit, die hier nur simuliert wird. Wichtig ist, dass Sie innerhalb dieser Methode threadsicher über den `Dispatcher` der Steuerelemente auf diese zugreifen.

Der dann folgende Delegat für das Callback-Ereignis wird ähnlich erzeugt, lediglich mit dem Unterschied, dass er dem `AsyncCallback`-Delegaten entsprechen muss, der ein Argument vom Typ `IAsyncResult` besitzt.

Schließlich wird die Methode, die der Delegat *worker* referenziert, über `BeginInvoke` asynchron ausgeführt, wobei der Callback-Delegat übergeben wird. Am letzten Argument der `BeginInvoke`-Methode übergebe ich `null`. Dieses Argument ist für Parameter vorgesehen, die beim asynchronen Aufruf an die Methode und den Ende-Delegaten durchgereicht werden.

Hinweis

Eine Einführung zu Lambda-Ausdrücken finden Sie in dem Artikel »Lambda-Ausdrücke und Ausdrucksbäume« auf der Buch-CD im Ordner »Zusatz-Artikel«.

270 Splash-Fenster

Splash-Fenster sind Fenster, die beim Starten einer Anwendung erscheinen, um dem Anwender während einer länger andauernden Initialisierungsphase Informationen anzubieten oder um einfach nur anzuzeigen, dass das Programm im Moment initialisiert wird.

Ein vernünftiges Splash-Fenster ist unter WPF aber erstaunlicherweise gar nicht so einfach zu implementieren:

- Wenn Sie die Initialisierung im `Loaded`-Ereignis des Startfensters der Anwendung vornehmen, können Sie das Splash-Fenster in diesem Ereignis zwar erzeugen, anzeigen und während der Initialisierung aktualisieren. Das Problem dabei ist aber, dass das Hauptfenster im `Loaded`-Ereignis bereits angezeigt wird, was in der Regel unerwünscht ist. Eine Lösung des Problems wäre, die Initialisierung im Konstruktor des Hauptfensters vorzunehmen.
- Wenn Sie die Initialisierung im `Startup`-Ereignis der Anwendung ausführen und das Splash-Fenster dort anzeigen, während die Initialisierung läuft, und danach schließen, wird das Hauptfenster der Anwendung (das in der `StartupUri`-Eigenschaft angegeben ist) nicht angezeigt, wenn die Eigenschaft `ShutdownMode` der Anwendung nicht auf `OnExplicitShutdown` steht. Eine Lösung dieses Problems ist natürlich, `ShutdownMode` auf `OnExplicitShutdown` zu setzen. Dann müssen Sie allerdings zum Beenden der Anwendung explizit die `Shutdown`-Methode aufrufen (was Sie idealerweise im `Closed`-Ereignis des Hauptfensters programmieren).
- Ein anderes Problem ist, dass das Splash-Fenster Informationen, die Sie in dieses schreiben (z.B. in einen `TextBlock` und eine `ProgressBar` eine Information über den Fortschritt), während der Initialisierung nicht anzeigt, weil Windows die Aktualisierung eines Fensters nicht ausführt, wenn gleichzeitig prozessorlastige Aktionen laufen. Dieses Problem können Sie unter WPF leider nur lösen, indem Sie die Initialisierung in einem separaten Thread oder in einer asynchron aufgerufenen Methode ausführen.

Eine funktionierende (aber mit Sicherheit nicht einzige) Lösung basiert darauf, dass Sie in der *App.xaml*-Datei kein Startfenster angeben. Die Initialisierung der Anwendung nehmen Sie im `Startup`-Ereignis vor, in dem Sie auch das Splash-Fenster erzeugen und während der Initialisierung anzeigen. Im einfachsten Fall, ohne Aktualisierung des Splash-Fensters, sieht das Ganze folgendermaßen aus:

```
public partial class App : Application
{
   protected override void OnStartup(StartupEventArgs e)
   {
      base.OnStartup(e);
```

Listing 519: Einfaches Anzeigen eines statischen Splash-Fensters

```
      // Splash-Fenster öffnen
      SplashWindow splashWindow = new SplashWindow();
      splashWindow.Show();

      // Die Initialisierung ausführen
      for (int i = 0; i < 100; i++)
      {
         // Die Initialisierung wird hier nur simuliert
          System.Threading.Thread.Sleep(30);
      }

      // Das Hauptfenster erzeugen, der Anwendung zuweisen
      // und öffnen
      MainWindow mainWindow = new MainWindow();
      this.MainWindow = mainWindow;
      mainWindow.Show();

      // Schließen des Splash-Fensters
      splashWindow.Close();
   }
}
```

Listing 519: Einfaches Anzeigen eines statischen Splash-Fensters (Forts.)

Ein auf diese Weise angezeigtes Splash-Fenster kann aber keine Informationen anzeigen, die während der Initialisierung ausgegeben werden sollen. Außerdem werden Animationen, die auf dem Splash-Fenster angelegt sind, nicht ausgeführt.

Die Lösung dieser Probleme setzt eine asynchrone Ausführung der Initialisierung ein. Ich verwende in dem folgenden Beispiel ein Splash-Fenster, das folgendermaßen definiert ist:

```
<Window x:Class="Splash_Fenster.SplashWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="180" Width="280"
    WindowStyle="None">
   <Grid>
      <TextBlock Name="txtInfo" Margin="10,10,0,28" TextWrapping="Wrap"
         HorizontalAlignment="Left" Width="166">Info</TextBlock>
      <ProgressBar Name="pbr" Height="20" VerticalAlignment="Bottom"/>
   </Grid>
</Window>
```

Listing 520: Das Splash-Fenster

Die partielle C#-Klasse des Fensters enthält eine Methode zur Aktualisierung der Informationen des Fensters:

```
public void SetInfo(string info, int progress)
{
   this.txtInfo.Text = info;
   this.pbr.Value = progress;
}
```

Im Startup-Ereignis der Anwendung erfolgt die Initialisierung nun asynchron. Dazu ist zunächst ein Delegat notwendig, der später asynchron ausgeführt wird und der die Initialisierung (in Form eines Lambda-Ausdrucks) verwaltet:

```
public partial class App : Application
{
   protected override void OnStartup(StartupEventArgs e)
   {
      base.OnStartup(e);

      // Splash-Fenster öffnen
      SplashWindow splashWindow = new SplashWindow();
      splashWindow.Show();

      // Delegat für die asynchrone Ausführung der Initialisierung
      Action initializer = new Action(() =>
      {
         // Simulation einer Initialisierung
         try
         {
            /* ******* Initialisierung ******** */
            for (int i = 0; i < 100; i++)
            {
               // Das Splash-Fenster aktualisieren
               splashWindow.Dispatcher.Invoke(DispatcherPriority.Normal,
                   new Action(() =>
                   {
                      splashWindow.SetInfo("Lese Datensatz " + i + " ...", i);
                   }));

               // Die eigentliche Initialisierung wird hier nur simuliert
               System.Threading.Thread.Sleep(30);
            }
            /* *** Ende der Initialisierung *** */
         }
         catch (Exception ex)
         {
            MessageBox.Show("Fehler beim Initialisieren: " + ex.Message,
               "Splash-Fenster", MessageBoxButton.OK, MessageBoxImage.Error);
```

Listing 521: Öffnen des Splash-Fensters und Delegat für die Initialisierung

```
            this.Shutdown();
         }
      });
```

Listing 521: Öffnen des Splash-Fensters und Delegat für die Initialisierung (Forts.)

Danach folgt ein Delegat, der als Callback für das Ende der asynchronen Ausführung verwendet wird. Die diesem Delegaten zugewiesene Methode erzeugt das Hauptfenster der Anwendung, übergibt eine Referenz darauf an die `MainWindow`-Eigenschaft der Anwendung und öffnet das Fenster. Erst danach wird das Splash-Fenster geschlossen. Würde dieses vor dem Öffnen des Hauptfensters geschlossen, würde die Anwendung sich damit automatisch beenden (wenn `ShutdownMode` nicht auf `OnExplicitShutdown` steht):

```
      AsyncCallback splashCloser = new AsyncCallback((result) =>
      {
         // Das Hauptfenster erzeugen, der Anwendung zuweisen
         // und öffnen
         this.Dispatcher.Invoke(DispatcherPriority.Normal,
            new Action(() =>
            {
               MainWindow mainWindow = new MainWindow();
               this.MainWindow = mainWindow;
               mainWindow.Show();
            }));

         // Schließen des Splash-Fensters
         splashWindow.Dispatcher.Invoke(DispatcherPriority.Normal,
            new Action(() =>
            {
               splashWindow.Close();
            }));
      });
```

Listing 522: Delegat für das Öffnen des Hauptfensters und das Schließen des Splash-Fensters

Schließlich wird nur noch die asynchrone Ausführung gestartet:

```
      initializer.BeginInvoke(splashCloser, null);
   }
}
```

Listing 523: Asynchrones Starten der Initialisierung

Diese Lösung ist zwar aufwändig, sichert aber zum einen ab, dass Splash-Fenster dynamische Informationen anzeigen können, und erlaubt zum anderen auch Animationen auf den Splash-Fenstern.

271 Die aktuelle DPI-Einstellung der Bildschirme des Systems ermitteln

WPF setzt zur Positionierung und Größeneinstellung die geräteunabhängige Einheit 1/96 Zoll ein. Ein Standardbildschirm besitzt normalerweise einen DPI-Wert (Dots Per Inch = Punkte pro Zoll) von 96. Eine Einheit in WPF entspricht also normalerweise einem Punkt auf dem Bildschirm.

Bildschirme können aber auch einen anderen physikalischen DPI-Wert aufweisen. Bei Notebooks und 19-Zoll-TFT-Bildschirmen ist dies z.B. fast immer der Fall. Ein Notebook mit einer Monitor-Breite (nicht Diagonale!) von 13 Zoll und einer Auflösung von 1680 * 1024 Pixeln hat z.B. einen physikalischen DPI-Wert von ca. 129. Dieser DPI-Wert muss in Windows eingestellt sein, damit alles in einer korrekten Größe angezeigt wird. In der Voreinstellung ist Windows aber auf 96 DPI eingestellt, weswegen Notebooks Fenster, Steuerelemente, Schriften etc. kleiner anzeigen und 19-Zoll-Bildschirme (mit einer Auflösung von 1280 * 1024 Zoll, die dieselbe ist wie bei 17-Zoll-Monitoren), größer. Diese Einstellung finden Sie übrigens in den Eigenschaften des Desktop.

Wenn Sie nun direkt mit den Bildschirmen eines Systems arbeiten wollen (wie im Rezept 272), müssen Sie die Bildschirm-Pixel in die geräteunabhängige Einheit von WPF umrechnen. Und dafür benötigen Sie die aktuelle DPI-Einstellung.

Diese erhalten Sie über einen kleinen Trick über ein `PresentationSource`-Objekt. Ein solches Objekt wird für die Interoperabilität zwischen WPF und anderen Systemen verwendet. Eines dieser anderen Systeme ist z.B. der Monitor, der ggf. eine spezielle DPI-Einstellung verwenden kann.

Ein `PresentationSource`-Objekt erhalten Sie über die `FromVisual`-Methode der `PresentationSource`-Klasse, der Sie ein WPF-Element übergeben (das von `Visual` abgeleitet sein muss, was bei Steuerelementen und Fenstern der Fall ist). Die Eigenschaft `CompositionTarget` liefert Informationen zur Anzeigeoberfläche des Elements, für das die `PresentationSource`-Instanz erzeugt wurde. Die `TransformToDevice`-Eigenschaft liefert ein `Matrix`-Objekt, über das das WPF-Element zum Ausgabegerät transformiert wird. Eine solche Matrix verwende ich in Rezept 331. Ich will hier deshalb nicht näher darauf eingehen. Die Eigenschaft `M11` (die für den Wert am Matrix-Index 1,1 steht) verwaltet hier den Faktor, mit dem die geräteunabhängige Einheit multipliziert werden muss, um die entsprechende Anzahl Pixel in der Horizontalen zu ergeben. Für `M22` gilt Ähnliches, nur für die Vertikale. Aber sei's drum: Es funktioniert.

So können Sie die Umrechenfaktoren und damit auch den aktuellen DPI-Wert berechnen:

```
double pixelCalculationFactorX = 1;
double pixelCalculationFactorY = 1;
PresentationSource source = PresentationSource.FromVisual(this);
if (source != null)
{
   pixelCalculationFactorX = source.CompositionTarget.TransformToDevice.M11;
```

Listing 524: Ermittlung der Faktoren zur Umrechnung der WPF-Einheit in Pixel und der aktuellen DPI-Einstellung

```
    pixelCalculationFactorY = source.CompositionTarget.TransformToDevice.M22;
}
double dpiX = 96 * pixelCalculationFactorX;
double dpiY = 96 * pixelCalculationFactorY;
```

Listing 524: Ermittlung der Faktoren zur Umrechnung der WPF-Einheit in Pixel und der aktuellen DPI-Einstellung (Forts.)

Das Beispiel erfordert den Import des Namensraums `System.Windows`.

272 Ein Fenster auf einem sekundären Bildschirm öffnen

In einer WPF-Anwendung werden Fenster standardmäßig auf dem primären Bildschirm geöffnet. Lediglich wenn ein Fenster mit einem Besitzer geöffnet wird, wird dieses auf dem Bildschirm geöffnet, auf dem der größte Teil des Besitzer-Fensters liegt. In einigen Fällen wollen Sie aber vielleicht ein Fenster explizit auf einem sekundären Bildschirm öffnen (natürlich nur, sofern das System über mehrere Bildschirme verfügt).

Ein Problem bei der Suche nach einer Lösung dieses Problems ist, dass WPF leider (noch) keine Informationen über die sekundären Bildschirme des Systems zur Verfügung stellt. Über `SystemParameters.PrimaryScreenWidth` und `SystemParameters.PrimaryScreenHeight` erhalten Sie lediglich die Breite und Höhe des primären Bildschirms.

Dieses Problem können Sie pragmatisch (was meint ohne API-Funktionen ☺) lösen, indem Sie sich bei `Windows.Forms` die `Screen`-Klasse ausborgen. Über deren `AllScreens`-Array erhalten Sie Informationen zu den Bildschirmen des Systems.

Das zweite Problem ist dann aber, dass WPF die geräteunabhängige Einheit 1/96 Zoll verwendet, `Screen` die Bildschirmposition und -größe aber in Pixeln angibt. Auf einem Standardbildschirm mit 96 DPI (oder PPI = Points Per Inch) macht das keinen Unterschied. Wenn es sich aber um einen Bildschirm mit einer anderen Auflösung handelt (die in Windows auch so eingestellt ist, was bei vielen Notebooks z.B. *nicht* der Fall ist), stimmen die WPF-Angaben nicht mit den `Screen`-Angaben überein.

Dieses Problem habe ich so gelöst, dass ich über Rezept 271 den Faktor für die Umrechnung in die aktuelle DPI-Einstellung auslese und diesen zur Umrechnung der berechneten Werte verwende.

Die folgende Methode `SetPositionToFirstSecondaryScreen` stellt auf diese Weise die Position eines Fensters auf die Mitte des ersten sekundären Bildschirms ein. Ist nur der primäre Bildschirm vorhanden, wird das Fenster auf die Mitte dieses Bildschirms eingestellt. `SetPositionToFirstSecondaryScreen` erwartet neben der Übergabe des einzustellenden Fensters auch ein Parent-Fenster. Dieses wird benötigt, um die DPI-Einstellung ermitteln zu können (`PresentationSource.FromVisual` ergibt für ein noch nicht sichtbares Fenster `null`, bei dem einzustellenden Fenster ist aber davon auszugehen, dass dieses gerade erst erzeugt, aber noch nicht angezeigt wurde).

Zum Kompilieren müssen Sie die Assemblys *System.Windows.Forms.dll* und *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Windows` und `System.Windows.Forms` einbinden.

```
public static void SetPositionToFirstSecondaryScreen(Window window,
   Window parentWindow)
{
   // Die Argumente überprüfen
   if (window == null)
   {
      throw new ArgumentException("window darf nicht null sein");
   }
   if (parentWindow == null)
   {
      throw new ArgumentException("parentWindow darf nicht null sein");
   }

   // Start-Position auf manuell stellen
   window.WindowStartupLocation = WindowStartupLocation.Manual;

   // Alle Bildschirme des Systems durchgehen
   // um den ersten sekundären zu finden
   Screen screen = Screen.PrimaryScreen;
   if (Screen.AllScreens.Length > 1)
   {
      // Sekundär-Bildschirme sind vorhanden: Suchen des ersten
      // Sekundär-Bildschirms
      for (int i = 0; i < Screen.AllScreens.Length; i++)
      {
         // Überprüfen, ob der aktuelle Bildschirm nicht der primäre ist
         if (Screen.AllScreens[i] != Screen.PrimaryScreen)
         {
            screen = Screen.AllScreens[i];
            break;
         }
      }
   }

   // Den Faktor für die Umrechnung von Pixeln in die geräteunabhängige
   // Einheit (DIU = Device Independent Unit) von WPF ermitteln
   double diuCalculationFactorX = 1;
   double diuCalculationFactorY = 1;
   PresentationSource source = PresentationSource.FromVisual(parentWindow);
   if (source != null)
```

Listing 525: Methode zum Platzieren eines Fensters auf dem ersten sekundären Bildschirm des Systems unter Berücksichtigung der aktuellen DPI-Einstellung

```
    {
        diuCalculationFactorX =
            1 / source.CompositionTarget.TransformToDevice.M11;
        diuCalculationFactorY =
            1 / source.CompositionTarget.TransformToDevice.M22;
    }
    else
    {
        throw new ArgumentException("Für parentWindow kann kein " +
            "PresentationSource-Objekt erzeugt werden. parentWindow " +
            "muss ein existierendes und angezeigtes Fenster sein.");
    }

    // Die Position und Größe des Bildschirms in die geräteunabhängige
    // Einheit von WPF (1/96 Zoll) umrechnen
    double screenLeft = screen.WorkingArea.Left * diuCalculationFactorX ;
    double screenTop = screen.WorkingArea.Top * diuCalculationFactorY ;
    double screenWidth = screen.WorkingArea.Width * diuCalculationFactorX;
    double screenHeight = screen.WorkingArea.Height * diuCalculationFactorY ;

    // Fenster auf den gefundenen, evtl. sekundären Bildschirm platzieren
    window.Left = screenLeft + ((screenWidth - window.Width) / 2);
    window.Top = screenTop + ((screenHeight - window.Height) / 2);
}
```

Listing 525: Methode zum Platzieren eines Fensters auf dem ersten sekundären Bildschirm des Systems unter Berücksichtigung der aktuellen DPI-Einstellung (Forts.)

273 Das Hauptfenster einer Anwendung ermitteln

Wenn Sie z.B. in einer Klassenbibliothek das Hauptfenster einer Anwendung ermitteln wollen, können Sie unter WPF einfach die `MainWindow`-Eigenschaft der aktuellen Anwendung auslesen:

```
using System.Windows;

...

Window mainWindow = Application.Current.MainWindow;
```

274 Die absolute und die Bildschirm-Position eines Steuerelements ermitteln

WPF verwaltete keine absolute Position für Steuerelemente (außer wenn diese auf einem `Canvas` platziert werden). Wenn Sie aber die absolute Position eines Steuerelements bezogen auf den Innenbereich des Fensters (oder der Seite) ermitteln wollen, auf dem das Steuerelement angelegt ist, können Sie die `TranslatePoint`-Methode des Elements verwenden. Am ersten Argument übergeben Sie einen Punkt, der einen Offset angibt. Wenn Sie die absolute Position ermitteln wollen, übergeben Sie hier einen Punkt, der mit (0, 0) initialisiert ist. Am zweiten

Argument übergeben Sie eine Referenz auf das UI-Element, zu dem die Position relativ berechnet werden soll. Für unseren Fall ist das das Fenster (oder die Seite), auf dem das Steuerelement angelegt ist:

```
using System;
using System.Windows;

...

// Das Element, dessen absolute Position ermittelt werden soll
UIElement uiElement = ...;

// Das Fenster, auf das die Position bezogen werden soll
Window window = ...;

// Die absolute Position bezogen auf das Fenster ermitteln
Point absolutePosition = uiElement.TranslatePoint(
   new Point(0, 0), window);
```

Listing 526: Ermitteln der absoluten Position eines UI-Elements

In einigen Fällen benötigen Sie aber auch die auf den Bildschirm bezogene Position des UI-Elements, z.B. um an dieser Position (oder relativ dazu) ein Fenster öffnen zu können. Die Bildschirm-Position erhalten Sie über die `PointToScreen`-Methode des Fensters (oder der Seite), auf dem das Steuerelement liegt:

```
Point screenPosition = window.PointToScreen(absolutePosition);
```

Listing 527: Die Bildschirm-Position eines UI-Elements ermitteln

Hinweis

Das Ergebnis von `PointToScreen` ist ein Punkt in der Einheit des Bildschirms, also in der Regel in Pixel, nicht in der geräteunabhängigen Einheit von WPF (die von Microsoft-Mitarbeitern als DIU bezeichnet wird = Device Independent Unit)! Falls Sie die DIU-Einheit benötigen (z. B. um ein Fenster positionieren zu können), müssen Sie den Punkt umrechnen:

```
double diuCalculationFactorX = 1;
double diuCalculationFactorY = 1;
PresentationSource source = PresentationSource.FromVisual(window);
if (source != null)
{
   diuCalculationFactorX =
      1 / source.CompositionTarget.TransformToDevice.M11;
```

Listing 528: Umrechnen der Pixel-Position in die DIU-Einheit

```
   diuCalculationFactorY =
      1 / source.CompositionTarget.TransformToDevice.M22;
}
else
{
   throw new ArgumentException("Für window kann kein " +
      "PresentationSource-Objekt erzeugt werden. window " +
      "muss ein existierendes und angezeigtes Fenster sein.");
}
Point diuScreenPosition = new Point(
   screenPosition.X * diuCalculationFactorX,
   screenPosition.Y * diuCalculationFactorY);
```

Listing 528: Umrechnen der Pixel-Position in die DIU-Einheit (Forts.)

275 Die optimale Position eines Fensters bezogen auf ein Steuerelement ermitteln

In einem `Windows.Forms`-Projekt musste ich einmal die Position eines Fensters (ok, in `Windows.Forms` heißt das »eines Formulars« ...) so berechnen, dass dieses in der Nähe eines Schalters lag, der das Fenster öffnete. Das Problem dabei war, dass der Schalter auf dem Bildschirm auch so liegen konnte, dass das Fenster nicht komplett angezeigt werden würde, wenn es einfach nur unterhalb des Schalters geöffnet würde. Also habe ich die Methode `DockToControl` entwickelt, die ein Fenster (in der `Windows.Forms`-Version in Rezept 235 eigentlich ein beliebiges Steuerelement, aber das ist unter WPF nicht möglich) so positioniert, dass es in der Nähe eines Steuerelements liegt, aber komplett angezeigt wird. In Listing 529 finden Sie die für WPF umgesetzte Variante, die auf dem Rezept 274 basiert. In Rezept 235 finden Sie eine nähere Erläuterung dieser Methode (für `Windows.Forms`).

`DockToControl` erwartet am ersten Argument eine Referenz auf das Steuerelement, an dem das Fenster ausgerichtet werden soll, und am zweiten eine Referenz auf das zu positionierende Fenster. Zum Kompilieren dieser Methode müssen Sie neben den unter WPF üblichen Assemblys (wegen der Verwendung der `Screen`-Klasse) die Assemblys *System.Windows.Forms.dll* und *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Windows` und `System.Windows.Controls` einbinden.

```
public static void DockToControl(Control referenceControl, Window window)
{
   // Die Argumente überprüfen
   if (referenceControl == null)
   {
      throw new ArgumentNullException("referenceControl darf nicht " +
         "null sein");
   }
   if (window == null)
```

Listing 529: Methode zum Andocken eines Fensters an ein Steuerelement

```
{
   throw new ArgumentNullException("window darf nicht null sein");
}

// Das Fenster des Referenz-Steuerelements suchen
Window parentWindow = null;
FrameworkElement temp = referenceControl;
while (temp != null)
{
   if (temp is Window)
   {
      parentWindow = (Window)temp;
      break;
   }
   if (temp.Parent is FrameworkElement)
   {
      temp = (FrameworkElement)temp.Parent;
   }
   else
   {
      break;
   }
}
if (parentWindow == null)
{
   throw new ArgumentException("Konnte kein Fenster für das " +
      "übergebene Steuerelement finden");
}

// Die auf den Bildschirm bezogene Position des Referenz-
// Steuerelements ermitteln.
Point absolutePositionOfReferenceControl =
   referenceControl.TranslatePoint(new Point(0, 0), parentWindow);
Point screenPositionOfReferenceControl =
   parentWindow.PointToScreen(absolutePositionOfReferenceControl);

// Ermitteln, auf welchem Bildschirm die X-Position
// des Steuerelements liegt
System.Windows.Forms.Screen[] screens =
   (System.Windows.Forms.Screen[])
   System.Windows.Forms.Screen.AllScreens.Clone();
Array.Sort(screens, (x, y) => x.Bounds.Left.CompareTo(y.Bounds.Left));
System.Windows.Forms.Screen controlScreen = screens[0];
foreach (var screen in screens)
{
```

Listing 529: Methode zum Andocken eines Fensters an ein Steuerelement (Forts.)

```
      if (screenPositionOfReferenceControl.X >= screen.Bounds.Left &&
         screenPositionOfReferenceControl.X <= screen.Bounds.Left +
         screen.Bounds.Width)
      {
         controlScreen = screen;
         break;
      }
   }

   // Den Faktor für die Umrechnung von Pixeln in die geräteunabhängige
   // Einheit (DIU = Device Independent Unit) von WPF ermitteln
   double diuCalculationFactorX = 1;
   double diuCalculationFactorY = 1;
   PresentationSource source = PresentationSource.FromVisual(parentWindow);
   if (source != null)
   {
      diuCalculationFactorX =
         1 / source.CompositionTarget.TransformToDevice.M11;
      diuCalculationFactorY =
         1 / source.CompositionTarget.TransformToDevice.M22;
   }
   else
   {
      throw new ArgumentException("Für parentWindow kann kein " +
         "PresentationSource-Objekt erzeugt werden. parentWindow " +
         "muss ein existierendes und angezeigtes Fenster sein.");
   }

   // Die Breite und Höhe des Fensters und Referenz-Steuerelements
   // in Pixeln berechnen
   double windowPixelWidth = window.Width / diuCalculationFactorX;
   double windowPixelHeight = window.Height / diuCalculationFactorY;
   double referenceControlPixelWidth =
      referenceControl.Width / diuCalculationFactorX;
   double referenceControlPixelHeight =
      referenceControl.Height / diuCalculationFactorY;

   // Basis-Position setzen
   Point location = new Point(screenPositionOfReferenceControl.X,
      screenPositionOfReferenceControl.Y);

   // Ermitteln, ob das Fenster nach unten noch passt
   if (location.Y + referenceControl.Height + windowPixelHeight <=
      controlScreen.WorkingArea.Bottom)
   {
```

Listing 529: Methode zum Andocken eines Fensters an ein Steuerelement (Forts.)

```
            location.Y += referenceControl.Height + 1;
        }
        else
        {
            location.Y = location.Y - windowPixelHeight;
        }

        // Ermitteln, ob das Fenster nach rechts noch passt
        if (location.X + referenceControlPixelWidth + windowPixelWidth >
            controlScreen.WorkingArea.Right)
        {
            location.X = screenPositionOfReferenceControl.X +
                referenceControlPixelWidth - windowPixelWidth;
            if (location.X + windowPixelWidth > controlScreen.WorkingArea.Right)
            {
                location.X = controlScreen.Bounds.Right - windowPixelWidth;
            }
        }

        // Ermitteln, ob das Fenster nach links noch passt
        if (location.X < controlScreen.WorkingArea.Left)
        {
            location.X = controlScreen.WorkingArea.Left;
        }
        // Ermitteln, ob das Fenster nach unten noch passt
        if (location.Y + window.Height > controlScreen.WorkingArea.Bottom)
        {
            location.Y = controlScreen.WorkingArea.Bottom - windowPixelHeight;
        }

        // Die Position des Fensters setzen
        window.Left = location.X * diuCalculationFactorX;
        window.Top = location.Y * diuCalculationFactorY;
    }
```

Listing 529: Methode zum Andocken eines Fensters an ein Steuerelement (Forts.)

Hinweis

Der einzige Punkt dieser Methode, bei dem ich nicht ganz sicher bin, ist die Suche nach dem Fenster des Steuerelements. Das Problem ist, dass der `Parent` eines `FrameworkElement` vom Typ `DependencyObject` ist. `DependencyObject` selbst besitzt aber keine `Parent`-Eigenschaft. Ist ein nicht von `FrameworkElement` abgeleitetes Objekt eines der Parent-Objekte, schlägt die Suche nach dem Fenster fehl. Das kann z.B. der Fall sein, wenn das Referenz-Steuerelement in einem `GridViewColumn`-Element angelegt ist.

276 Beim Öffnen eines Fensters den Fokus setzen

Ein WPF-Fenster (oder eine Seite) setzt beim Öffnen den Eingabefokus nicht auf ein Steuerelement. Dieses Verhalten ist beabsichtigt. So kann der Entwickler entscheiden, ob er den Fokus auf ein Steuerelement setzen will oder nicht.

In den meisten Anwendungen macht es aber Sinn, dass beim Öffnen eines Fensters der Fokus auf dem ersten Steuerelement in der Tabulator-Reihenfolge liegt. Dieses Problem können Sie recht einfach lösen, indem Sie im Ereignishandler des `Loaded`-Ereignisses des Fensters die folgende Anweisung unterbringen:

```
this.MoveFocus(new TraversalRequest(FocusNavigationDirection.Next));
```

Diese Anweisung führt dazu, dass der Fokus auf das nächste Steuerelement in der Tabulator-Reihenfolge gesetzt wird, simuliert also im Prinzip die Betätigung der TAB-Taste.

277 Das Einfügen über die Zwischenablage abfangen

Wenn Sie das Einfügen in ein Steuerelement über die Zwischenablage abfangen wollen, können Sie unter WPF der `DataObject`-Klasse einen Einfüge-Handler hinzufügen. Dazu rufen Sie die statische Methode `AddPastingHandler` auf und übergeben am ersten Argument das Steuerelement, für das Sie das Einfügen überwachen wollen. Am zweiten Argument übergeben Sie einen Delegaten vom Typ `DataObjectPastingEventHandler`, der die folgende Signatur aufweist:

```
void Name(object sender, DataObjectPastingEventArgs e)
```

Die Zwischenablagedaten ermitteln Sie dann wie üblich über die `Clipboard`-Klasse. Wenn Sie das Einfügen abbrechen wollen, rufen Sie die Methode `CancelCommand` des Ereignisargument-Objekts auf.

Das folgende (relativ sinnlose) Beispiel zeigt, wie dies prinzipiell programmiert wird. Es überprüft, ob die Zwischenablage Text enthält, vergleicht diesen mit dem String »C#« und lässt nur diesen String zum Einfügen über die Zwischenablage zu. Dieses Beispiel können Sie in einem Fenster einsetzen. Es erfordert den Import des Namensraums `System.Windows`.

```
/* Konstruktor. Initialisiert das Fenster. */
public MainWindow()
{
   InitializeComponent();

   // Den Ereignishandler für das Einfügen über die
   // Zwischenablage anfügen
   DataObject.AddPastingHandler(this.demoTextBox,
      new DataObjectPastingEventHandler(demoTextBox_Pasting));
}
```

Listing 530: Beispiel für das Abfangen des Einfügens über die Zwischenablage

```
/* Wird aufgerufen, wenn in die TextBox aus der Zwischenablage
   eingefügt werden soll */
private void demoTextBox_Pasting(object sender, DataObjectPastingEventArgs e)
{
   IDataObject dataObject = Clipboard.GetDataObject();
   if (dataObject.GetDataPresent(DataFormats.Text))
   {
      // Text aus der Zwischenablage auslesen
      string clipboardText =
         dataObject.GetData(DataFormats.Text).ToString();

      // Text überprüfen
      if (clipboardText != "C#")
      {
         // Das Einfügen abbrechen
         e.CancelCommand();
      }
   }
}
```

Listing 530: Beispiel für das Abfangen des Einfügens über die Zwischenablage (Forts.)

278 TextBox-Inhalt beim Eintritt komplett selektieren

Eine `TextBox`-Instanz verwaltet ihre aktuelle Selektion und stellt diese wieder her, wenn der Eingabefokus in das Steuerelement wechselt. In vielen Fällen soll aber beim Eintritt der gesamte enthaltene Text markiert werden. Dieses Problem können Sie in WPF auf verschiedene Weisen lösen:

- Über eine Methode, die Sie dem `GotFocus`-Ereignis zuweisen und die die `SelectAll`-Methode aufruft,
- über eine eigene von `TextBox` abgeleitete Klasse, die dieses Ereignis implizit abfängt und `SelectAll` aufruft,
- oder (besser) über einen Ereignis-Setter, der beim Fokuserhalt die Selektion setzt.

Die ersten beiden Lösungen habe ich bereits im entsprechenden `Windows.Forms`-Rezept gezeigt. Die Lösung über einen Ereignis-Setter ist in meinen Augen die beste. Leider ist eine Lösung über einen Trigger nicht möglich, da die Eigenschaften `SelectionStart` und `SelectionLength` keine Abhängigkeitseigenschaften sind und deswegen nicht in einem Trigger gesetzt werden können. Ein Ereignis-Setter in einem Stil, der für die `TextBox`-Klasse definiert ist, funktioniert aber auch. Diesen habe ich in den Ressourcen des Fensters erzeugt und auf das `GotFocus`-Ereignis gelegt. Die Ereignisbehandlungsmethode habe ich gleich im XAML-Code implementiert, damit das Kopieren einfacher ist:

```
<Window.Resources>
   <Style TargetType="{x:Type TextBox}">
      <EventSetter Event="GotFocus" Handler="TextBox_GotFocus"/>
   </Style>
   <x:Code>
      <![CDATA[
      private void TextBox_GotFocus(object sender, RoutedEventArgs e)
      {
         ((TextBox)sender).SelectAll();
      }
      ]]>
   </x:Code>
</Window.Resources>
```

Listing 531: Stil für eine TextBox, die sich beim Eintritt selbst selektiert (in den Ressourcen eines Fensters)

Hinweis

Denken Sie daran, dass Sie den Stil auch den Ressourcen der Anwendung zuweisen können, um zu erreichen, dass er für alle `TextBox`-Instanzen der gesamten Anwendung gilt.

279 TextBox auf Zahleingaben beschränken

Der Wunsch, eine `TextBox` auf die Eingabe von Zahlen einzuschränken, ist in meinen Seminaren einer der am häufigsten genannten. Sie können dieses Problem eingeschränkt lösen, indem Sie das `PreviewTextInput`-Ereignis der `TextBox` behandeln. Prüfen Sie die in der Ereignisargument-Eigenschaft `Text` übergebene Eingabe auf gültige Eingaben und verwerfen Sie alle anderen Eingaben, indem Sie die `Handled`-Eigenschaft des Ereignisargument-Objekts auf `true` setzen. Diese Lösung ist aber nicht besonders gut, da Sie dies für alle `TextBox`-Instanzen wiederholen müssen (was global über einen `EventSetter` in einem Stil möglich wäre). Daneben ist die Programmierung nicht trivial, wenn Sie auch negative Zahlen und Dezimalzahlen ermöglichen und die Zwischenablage behandeln wollen.

Aus diesem Grunde hatte ich ursprünglich für `Windows.Forms` eine von `TextBox` abgeleitete Klasse entwickelt, die es ermöglicht, den Anwender auf die Eingabe von Zahlen einzuschränken. Diese `NumberTextBox` finden Sie in diesem Rezept für WPF umgesetzt wieder. Ich erspare mir hier weitere allgemeine Erläuterungen. Lesen Sie im entsprechenden Rezept im `Windows.Forms`-Kapitel nach.

Ein Problem, das ich speziell für WPF lösen musste, war das Abfangen des Einfügens über die Zwischenablage. Dieses habe ich in Rezept 277 beschrieben.

Zum Kompilieren der Klasse `NumberTextBox` und der verwendeten Aufzählungen benötigen Sie den Import der Namensräume `System`, `System.ComponentModel`, `System.Windows`, `System.Windows.Controls` und `System.Windows.Input`.

```
/* Aufzählung für den Typ einer numerischen Eingabe */
public enum NumberInputTypes
{
   /* Als Eingabe sind Zahlen mit Nachkommastellen möglich */
   Double,

   /* Als Eingabe sind Zahlen ohne Nachkommastellen möglich */
   Integer
}

/* Ereignisargumentklasse für das InvalidInput-Ereignis. */
public class InvalidInputEventArgs : EventArgs
{
   /* Die ungültige Eingabe */
   public string Input
   {
      private set;
      get;
   }

   /* Konstruktor */
   internal InvalidInputEventArgs(string input)
   {
      this.Input = input;
   }
}

/* Textbox, die auf Zahleingaben beschränkt ist.
   Die Eigenschaft InputType gibt an, welche Eingaben erlaubt sind. */
public class NumberTextBox : TextBox
{
   /* Gibt an, welche Eingaben die TextBox zulässt */
   [Category("Behavior")]
   [Description("Gibt an, welche Eingaben die TextBox zulässt")]
   public NumberInputTypes InputType
   {
      get;
      set;
   }

   /* Wird aufgerufen, wenn der Anwender ungültige Daten eingibt */
   public event EventHandler<InvalidInputEventArgs> InvalidInput;

   /* Konstruktor */
   public NumberTextBox()
```

Listing 532: Klasse für eine TextBox, die nur Zahleingaben zulässt

```
{
   // Den Ereignishandler für das Einfügen über die
   // Zwischenablage anfügen
   DataObject.AddPastingHandler(this,
      new DataObjectPastingEventHandler(DataObjectPasting));
}

/* Ruft das <see cref="InvalidInput"/>-Ereignis auf */
protected virtual void OnInvalidInput(string input)
{
   if (this.InvalidInput != null)
   {
      this.InvalidInput(this, new InvalidInputEventArgs(input));
   }
}

/* Überprüft, ob die übergebene Eingabe gültig ist */
private bool CheckInput(string input)
{
   switch (this.InputType)
   {
      case NumberInputTypes.Double:
         try
         {
            Convert.ToDouble(input);
            return true;
         }
         catch
         {
            this.OnInvalidInput(input);
            return false;
         }

      default:  // InputTypeEnum.Integer
         try
         {
            Convert.ToInt64(input);
            return true;
         }
         catch
         {
            this.OnInvalidInput(input);
            return false;
         }
   }
```

Listing 532: Klasse für eine TextBox, die nur Zahleingaben zulässt (Forts.)

```
}

/* Wird überschrieben, um die Eingabe überprüfen zu können */
protected override void OnPreviewTextInput(TextCompositionEventArgs e)
{
   // Den potenziell neuen Text zusammensetzen
   string newText = base.Text.Substring(0, base.SelectionStart)
      + e.Text + base.Text.Substring(
      base.SelectionStart + base.SelectionLength);

   if (this.CheckInput(newText) == false)
   {
      // Die Eingabe führt zu einem ungültigen Ergebnis,
      // also verwerfen
      e.Handled = true;
   }

   base.OnPreviewTextInput(e);
}

/* Wird aufgerufen, wenn in das Steuerelement eingefügt werden soll */
private void DataObjectPasting(object sender, DataObjectPastingEventArgs e)
{
   IDataObject dataObject = Clipboard.GetDataObject();
   if (dataObject.GetDataPresent(DataFormats.Text))
   {
      // Text aus der Zwischenablage auslesen
      string clipboardText =
         dataObject.GetData(DataFormats.Text).ToString();
       // Die potenziell neue Eingabe zusammensetzen
      string input = this.Text;
      if (this.SelectionLength > 0)
      {
         // Wenn gerade Text selektiert ist, wird der Inhalt
         // der Zwischenablage in die Selektion eingefügt
         if (this.SelectionStart == 0)
         {
            input = clipboardText +
               input.Substring(this.SelectionLength,
               input.Length - this.SelectionLength);
         }
         else
         {
            input = input.Substring(0, this.SelectionStart) +
               clipboardText + input.Substring(this.SelectionStart +
```

Listing 532: Klasse für eine TextBox, die nur Zahleingaben zulässt (Forts.)

```
                this.SelectionLength, input.Length -
                this.SelectionStart - this.SelectionLength);
            }
         }
         else
         {
            // Wenn kein Text selektiert ist, wird der Inhalt der
            // Zwischenablage an der Cursorposition eingefügt
            input = input.Substring(0, this.SelectionStart) +
               clipboardText + input.Substring(this.SelectionStart,
               input.Length - this.SelectionStart);
         }
          // Das potenzielle Ergebnis überprüfen
         if (this.CheckInput(input) == false)
         {
            // Das Einfügen abbrechen
            e.CancelCommand();
         }
      }
   }
}
```

Listing 532: Klasse für eine TextBox, die nur Zahleingaben zulässt (Forts.)

Für den Fall, dass Sie nicht wissen, wie Sie diese Klasse einsetzen:

```
<Window x:Class="Demo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    ...
    xmlns:Namensraum-Alias="clr-namespace:Namensraum der NumberTextBox" >
    <Grid>
      <Namensraum-Alias:NumberTextBox x:Name="demoNumberTextBox" ... />
   </Grid>
</Window>
```

Ersetzen Sie die kursiven Angaben durch die in Ihrem Fall passenden Werte. Wenn die Klasse in einem Projekt erreichbar ist, hilft Visual Studio bei der Namensraum-Deklaration, wenn Sie »xmlns:*Namensraum-Alias*=« (z. B. xmlns:codebook=) schreiben. Wenn die Klasse in einer separaten Assembly verwaltet wird, muss diese bei der Namensraum-Deklaration zusätzlich angegeben werden. Lesen Sie ggf. meine WPF-Einführung, die Sie auf der Buch-CD im Ordner *Zusatz-Artikel* finden.

Hinweis

Die `NumberTextBox` fängt das Schreiben in die `Text`-Eigenschaft nicht ab. Der Grund dafür ist, dass diese Eigenschaft nicht virtuell definiert ist und deswegen nicht überschrieben werden kann. Außerdem handelt es sich dabei um eine Abhängigkeitseigenschaft, was eine Neudefinition erschwert. Ich gehe davon aus, dass das Programm wissen sollte, was in `Text` geschrieben wird.

280 Das TextChanged-Ereignis bei der ComboBox abfangen

Die WPF-`ComboBox` besitzt in der ersten Version eigenartigerweise kein `TextChanged`-Ereignis. Da dieses aber in der Praxis benötigt wird, habe ich nach einer Lösung gesucht. Und diese auch gefunden ☺.

Eine `ComboBox` verwendet intern eine `TextBox`, wenn die Eigenschaft `IsEditable true` ist. Meine Lösung basiert nun auf einer von `ComboBox` abgeleiteten Klasse, die das `TextChanged`-Ereignis dieser `TextBox` abfängt und weitergibt. Dazu wird die `TextBox` über deren Namen (*PART_EditableTextBox*) im visuellen Baum ermittelt. Dieses Vorgehen ist ein wenig unsicher, da der visuelle Baum der WPF-Steuerelemente in neueren WPF-Versionen auch geändert werden kann. Dabei ist allerdings eher wahrscheinlich, dass der Name der `TextBox` geändert wird, als dass an deren Stelle ein komplett anderes Steuerelement eingesetzt wird. Für den Fall, dass die TextBox nicht ermittelt werden kann, schreibt `ExtComboBox` einen entsprechenden Eintrag in das Trace-Protokoll.

In neueren Versionen von WPF kann es natürlich auch sein, dass die `ComboBox` ein `TextChanged`-Ereignis besitzt (eigentlich ist das sogar sehr wahrscheinlich). In diesem Fall ist dieses Rezept natürlich überflüssig ...

```
public class ExtComboBox : ComboBox
{
   // Das geroutete Ereignis deklarieren
   public static readonly RoutedEvent TextChangedEvent =
      EventManager.RegisterRoutedEvent("TextChanged",
      RoutingStrategy.Bubble, typeof(RoutedEventHandler),
      typeof(ExtComboBox));

   /* Das eigene TextChanged-Ereignis */
   public event RoutedEventHandler TextChanged
   {
      add { this.AddHandler(ExtComboBox.TextChangedEvent, value); }
      remove { this.RemoveHandler(ExtComboBox.TextChangedEvent, value); }
   }

   /* Wird überschrieben, um die TextBox zu ermitteln und deren
      TextChanged-Ereignis zuzuweisen */
   protected override void OnRender(DrawingContext drawingContext)
   {
      base.OnRender(drawingContext);
      if (this.IsEditable)
      {
         // Die TextBox ist nur dann vorhanden, wenn IsEditable true ist:
         // Eine Referenz auf die TextBox ermitteln:
         TextBox textBox = this.GetTemplateChild(
            "PART_EditableTextBox") as TextBox;
```

Listing 533: Erweiterte ComboBox, die ein TextChanged-Ereignis zur Verfügung stellt

```
            // Der Name der TextBox (PART_EditableTextBox) kann sich in neuen
            // WPF-Versionen auch ändern. Deswegen wird auf null verglichen.
            if (textBox != null)
            {
               textBox.TextChanged +=
                  new TextChangedEventHandler(textBox_TextChanged);
            }
            else
            {
               // Eintrag in das Trace-Protokoll schreiben
               Trace.Write(this.GetType().Name + ": Keine TextBox mit dem " +
                  "Namen 'PART_EditableTextBox' gefunden");
            }
         }
      }

      /* Fängt das TextChanged-Ereignis der TextBox ab */
      private void textBox_TextChanged(object sender, TextChangedEventArgs e)
      {
         this.OnTextChanged(e);
      }

      /* Ruft das TextChanged-Ereignis auf */
      protected virtual void OnTextChanged(TextChangedEventArgs e)
      {
         RoutedEventArgs routedEventArgs =
            new RoutedEventArgs(ExtComboBox.TextChangedEvent);
         this.RaiseEvent(routedEventArgs);
      }
   }
```

Listing 533: Erweiterte ComboBox, die ein TextChanged-Ereignis zur Verfügung stellt (Forts.)

Für den Fall, dass Sie nicht wissen, wie Sie diese Klasse einsetzen:

```
<Window x:Class="Demo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    ...
    xmlns:Namensraum-Alias="clr-namespace:Namensraum der ExtComboBox" >
    <Grid>
      <Namensraum-Alias:ExtComboBox x:Name="demoComboBox"
         IsEditable="True" ...
         TextChanged="demoComboBox_TextChanged">
      </codebook:ExtComboBox>
   </Grid>
</Window>
```

Ersetzen Sie die kursiven Angaben durch die in Ihrem Fall passenden Werte. Wenn die Klasse in einem Projekt erreichbar ist, hilft Visual Studio bei der Namensraum-Deklaration, wenn Sie

»xmlns:*Namensraum-Alias*=« (z. B. xmlns:codebook=) schreiben. Wenn die Klasse in einer separaten Assembly verwaltet wird, muss diese bei der Namensraum-Deklaration zusätzlich angegeben werden. Lesen Sie ggf. meine WPF-Einführung, die Sie auf der Buch-CD im Ordner *Zusatz-Artikel* finden.

281 Bei der Betätigung der Return-Taste die Tab-Taste simulieren

Ein Wunsch vieler Anwender, die Massendaten eingeben müssen, ist, dass ein Programm bei der Betätigung der Return-Taste zum nächsten Steuerelemente in der Tabulator-Reihenfolge wechselt. Auch wenn dies mit modernen Windows-Standards nicht übereinstimmt, haben diese Anwender in vielen Fällen sogar Recht. Geben Sie einmal den ganzen Tag gleichförmige Daten in immer dieselben Formulare ein ...

Deswegen biete ich – wie auch schon im `Windows.Forms`-Kapitel – eine Lösung für dieses Problem.

Im Internet ist häufig zu diesem Problem zu finden, dass Sie dazu die Technik verwenden, die bereits in Rezept 276 dafür gesorgt hat, dass beim Öffnen eines Fensters der Eingabefokus auf dem ersten Steuerelement in der Tabulator-Reihenfolge liegt. Das funktionierte in meinen Tests aber eigenartigerweise für die Weitergabe der Return-Taste als Tab-Taste nicht (bzw. nicht immer).

Eine funktionierende Lösung setzt die `ProcessInput`-Methode der aktuellen `InputManager`-Instanz ein. Ein `InputManager` verwaltet alle Eingabesysteme von WPF. Über `InputManager.Current` erreichen Sie den aktuellen `InputManager`. Die `ProcessInput`-Methode verarbeitet eine »Eingabe«, die in Form einer `InputEventArgs`-Instanz übergeben wird. Um eine Taste zu simulieren übergeben Sie eine Instanz der `KeyEventArgs`-Klasse. Damit das Ganze korrekt funktioniert, müssen Sie noch das geroutete Ereignis `Keyboard.KeyDownEvent` in die Eigenschaft `RoutedEvent` schreiben und können dann schließlich `ProcessInput` aufrufen, um die Taste zu simulieren. Das Ganze macht Sinn im `KeyDown`-Ereignis der Steuerelemente, für die Sie die Return-Taste umsetzen wollen:

```
private void HandleKeyDown(object sender, KeyEventArgs e)
{
   if (e.Key == Key.Return)
   {
      // KeyEventArgs für die Simulation der Tab-Taste erzeugen
      KeyEventArgs keyEventArgs = new KeyEventArgs(
         Keyboard.PrimaryDevice, Keyboard.PrimaryDevice.ActiveSource,
         0, Key.Tab);

      // Simulieren, dass die Eingabe vom gerouteten Ereignis
      // KeyDownEvent stammt
      keyEventArgs.RoutedEvent = Keyboard.KeyDownEvent;
```

Listing 534: Umsetzen der Return-Taste in die Tab-Taste in einem Ereignishandler für das KeyDown-Ereignis

```
      // Die »Eingabe« verarbeiten
      InputManager.Current.ProcessInput(keyEventArgs);
   }
}
```

Listing 534: Umsetzen der Return-Taste in die Tab-Taste in einem Ereignishandler für das KeyDown-Ereignis (Forts.)

Alternativ können Sie auch einen Stil verwenden. Der folgende Stil, der in den Ressourcen eines Fensters angelegt ist, setzt die Problemlösung für die Klassen `TextBox`, `RichTextBox`, `PasswordBox`, `ListBox`, `ComboBox`, `CheckBox` und `RadioButton` um:

```
<Window.Resources>
   <Style TargetType="{x:Type TextBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type RichTextBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type PasswordBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type ListBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type ComboBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type CheckBox}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <Style TargetType="{x:Type RadioButton}">
      <EventSetter Event="KeyDown" Handler="Return2TabHandler" />
   </Style>
   <x:Code>
      <![CDATA[
      private void Return2TabHandler(object sender, KeyEventArgs e)
      {
         if (e.Key == Key.Return)
         {
            // KeyEventArgs für die Simulation der Tab-Taste erzeugen
            KeyEventArgs keyEventArgs = new KeyEventArgs(
               Keyboard.PrimaryDevice, Keyboard.PrimaryDevice.ActiveSource,
               0, Key.Tab);
```

Listing 535: Stil für die automatische Zuweisung des KeyDown-Ereignisses der Klassen TextBox, RichTextBox, PasswordBox, ListBox, ComboBox, CheckBox und RadioButton auf eine Methode, die die Return-Taste in eine Tab-Taste »umwandelt«

```
            // Simulieren, dass die Eingabe vom gerouteten Ereignis
            // KeyDownEvent stammt
            keyEventArgs.RoutedEvent = Keyboard.KeyDownEvent;

            // Die »Eingabe« verarbeiten
            InputManager.Current.ProcessInput(keyEventArgs);
         }
      }
      ]]>
   </x:Code>
</Window.Resources>
```

Listing 535: Stil für die automatische Zuweisung des KeyDown-Ereignisses der Klassen TextBox, RichTextBox, PasswordBox, ListBox, ComboBox, CheckBox und RadioButton auf eine Methode, die die Return-Taste in eine Tab-Taste »umwandelt« (Forts.)

Hinweis

Der Stil setzt Inline-Code ein, um zu erreichen, dass Sie den Stil einfacher kopieren können. Beachten Sie, dass Sie den Stil ggf. noch um Ereignis-Setter für weitere Steuerelemente (wie für `TreeView`) erweitern müssen. Bereits vorhandene Stile müssen Sie natürlich entsprechend anpassen. Beachten Sie auch, dass Sie den Stil statt in den Ressourcen eines Fensters in den Ressourcen der Anwendung anlegen können, um alle entsprechenden Steuerelemente der gesamten Anwendung mit der neuen Funktionalität auszustatten.

282 Drag&Drop von Dateien und Ordnern

Für Anwendungen, die Eingaben von Datei- oder Ordnernamen erwarten, ist es sinnvoll, Drag&Drop vom Explorer (oder von anderen Dateimanagern) aus zu ermöglichen. Die Lösung dieses Problems ist zwar relativ einfach, aber nicht unbedingt intuitiv. Weswegen dieses Rezept (genau wie sein `Windows.Forms`-Pendant) sicherlich gerechtfertigt ist.

Drag&Drop von Dateien und Ordnern ist unter WPF aber so nahezu identisch, dass ich mir eine Beschreibung der Vorgehensweise hier spare und auf das entsprechende Rezept im `Windows.Forms`-Kapitel verweise. Das folgende Beispiel setzt dieses Beispiel, das das Ziehen von Dateien auf eine `ListBox` erlaubt, für WPF um. Beachten Sie, dass die Eigenschaft `AllowDrop` der `ListBox` auf `true` stehen muss.

```
private void fileListBox_DragEnter(object sender, DragEventArgs e)
{
  // Überprüfen, ob Dateien oder Ordner gezogen werden
  if (e.Data.GetDataPresent(DataFormats.FileDrop))
  {
     e.Effects = DragDropEffects.Copy;
  }
  else
```

Listing 536: Drag&Drop von Dateien und Ordnern auf eine ListBox

```
  {
    e.Effects = DragDropEffects.None;
  }
}

private void fileListBox_Drop(object sender, DragEventArgs e)
{
  // Dateien aus den gezogenen Daten auslesen
  string[] filesNames = (string[])e.Data.GetData(DataFormats.FileDrop,
    false);
  foreach (string fileName in filesNames)
  {
    // FileInfo-Objekt erzeugen
    FileInfo fi = new FileInfo(fileName);
    if (fi.Exists)
    {
      // Wenn es sich nicht um einen Ordner handelt: FileInfo-Objekt der
      // Liste anfügen
      this.fileListBox.Items.Add(fi);
    }
  }

}
```

Listing 536: Drag&Drop von Dateien und Ordnern auf eine ListBox (Forts.)

283 In einem Nicht-Tastatur-Ereignis herausfinden, ob eine bestimmte Taste betätigt ist

In den Tastatur-Ereignissen erhalten Sie über das Ereignisargument-Objekt eine Information über die aktuell betätigten Tasten. Manchmal benötigen Sie aber auch an anderer Stelle im Programm eine Information darüber, ob bestimmte Tasten betätigt sind. Das kann z. B. in einem Mausereignis der Fall sein, wenn Sie feststellen wollen, ob der Anwender bei der Betätigung der Maus gleichzeitig eine Taste betätigt hat.

Unter WPF können Sie hier einfach die `Keyboard`-Klasse aus dem Namensraum `System.Windows.Input` einsetzen. Die statische Methode `IsKeyDown` liefert eine Information darüber, ob die in Form eines `Key`-Werts übergebene Taste gerade betätigt wird. Die statische Eigenschaft `Modifiers` liefert die zurzeit betätigten Modifiziertasten in Form der Werte der `ModifierKeys`-Aufzählung. So können Sie z. B. in einem `MouseLeftButtonDown`-Ereignis herausfinden, ob gleichzeitig mit dem Mausklick die Tastenkombination SHIFT + STRG + Q betätigt ist:

```
private void TextBlock_MouseLeftButtonDown(object sender,
   System.Windows.Input.MouseButtonEventArgs e)
{
   if ((Keyboard.Modifiers == (ModifierKeys.Shift | ModifierKeys.Control)) &&
      Keyboard.IsKeyDown(Key.Q))
   {
      ...
   }
}
```

Listing 537: Abfragen der Tastatur in einem Nicht-Tastatur-Ereignis

LINQ und LINQ to SQL

Dieses Kapitel enthält Rezepte für LINQ und LINQ to SQL. Grundlagen werden allerdings nicht behandelt. Lesen Sie ggf. meine Artikel »LINQ« und »LINQ to SQL«, die Sie auf der Buch-CD im Ordner *Zusatz-Artikel* finden.

Hinweis

Einige LINQ-to-SQL-Rezepte, die eher allgemeinen (Datenbank-)Charakter haben, finden Sie im Kapitel »Datenbank-Programmierung«.

284 Dynamische Abfragen

In der Praxis ist es häufig notwendig, Abfragen dynamisch aufzubauen. Bei der Abfrage einer Auflistung von Artikeldaten können Sie so z. B. dem Benutzer ermöglichen, nach beliebigen Kriterien wie dem Artikel-Namen, dem Preis oder der Kategorie zu suchen (Abbildung 163).

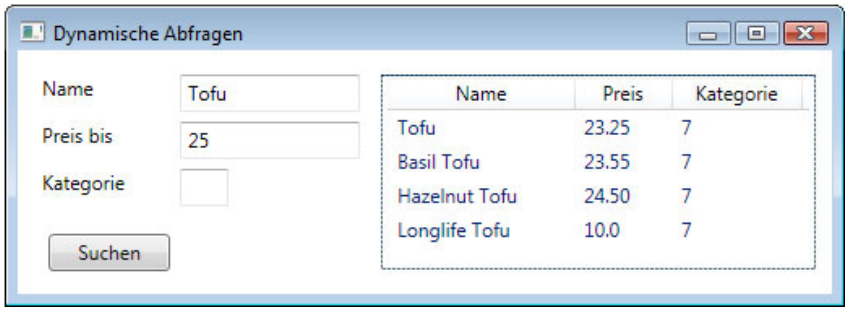


Abbildung 163: Beispiel für eine dynamische Abfrage

Das Problem, dass je nach Benutzereingabe unterschiedliche Kriterien abgefragt werden sollen, können Sie über zwei Techniken lösen: progressive Abfragen und dynamische Abfragen über einen Abfrage-String.

Zur Demonstration dieser Techniken verwende ich eine Auflistung von `Product`-Objekten, deren Klasse folgendermaßen definiert ist:

```
public class Product
{
   public string Name { get; set; }
   public decimal Price { get; set; }
   public int? CategoryID { get; set; }
}
```

Listing 538: Klasse für die Speicherung der Beispieldaten

Progressive Abfragen

Da eine LINQ-Abfrage, die eine Sequenz zurückgibt, immer eine `IEnumerable<T>`- oder `IQueryable<T>`-Referenz zurückgibt, können Sie mehrere Abfragen progressiv aufeinander aufbauen. Dabei verwenden Sie eine Variable für die Abfrage, die Sie ab der zweiten Abfrage als Sequenz benutzen. Auf diese Weise können Sie Abfragen schon relativ dynamisch aufbauen:

```
// Artikel einlesen
List<Product> products = GetProducts();

// Die Abfragekriterien zusammenstellen
string namePart = "Tofu";
decimal? maxPrice= 25;
int? categoryID = 1;

// Progressives Zusammenstellen der Abfrage
IEnumerable<Product> query = products;
if (String.IsNullOrEmpty(namePart ) == false)
{
   query = query.Where(product =>
      product.Name.IndexOf(namePart) > -1);
}
if (maxPrice!= null)
{
   query = query.Where(product =>
      product.Price <= minPrice.Value);
}
if (categoryID != null)
{
   query = query.Where(product =>
      product.CategoryID == categoryID .Value);
}

// Auswerten des Ergebnisses
int count = query.Count;
```

Listing 539: Progressiver Aufbau einer Abfrage

Die Performance ist unter LINQ direkt wahrscheinlich nicht besonders gut, da in diesem Fall mehrere Abfragen hintereinander ausgeführt werden. Unter LINQ-Erweiterungen wie LINQ to SQL sollte die Performance allerdings recht gut sein, weil alle Teil-Abfragen zunächst interpretiert und schließlich als ein einziger Befehl gegen die Datenquelle ausgeführt werden.

Beim progressiv-dynamischen Aufbau sind Sie lediglich bei der Verwendung der Operatoren eingeschränkt. Wenn Sie dem Anwender ermöglichen sollen, auch diese zu wählen, ist eine dynamische Abfrage über einen String wahrscheinlich geeigneter.

Dynamische Abfragen

Echte dynamische Abfragen werden in der Regel (und in diesem Rezept) über Strings definiert. Der String wird in der Laufzeit geladen oder zusammengestellt, in eine Abfrage konvertiert und diese ausgeführt. Damit sind Sie bei der Abfrage von Daten recht flexibel.

Sie brauchen bloß noch eine Möglichkeit, eine in einem String (mit einer entsprechenden Syntax) definierte dynamische Abfrage in eine LINQ-Abfrage zu konvertieren. Und diese Möglichkeit existiert in Form der Klassen der Datei *Dynamic.cs*, die Sie im Ordner *Samples\1033\CSharpSamples\LinqSamples\DynamicQuery\DynamicQuery* Ihrer Visual Studio-Installation finden (bzw. in den Beispielen, die Sie an der Adresse *code.msdn.microsoft.com/csharpsamples/Release/ProjectReleases.aspx* herunterladen können). Alternativ können Sie natürlich auch die Kopie verwenden, die das Beispiel zu diesem Rezept verwendet.

Diese Klassen erweitern die `IQueryable`-Schnittstelle (leider nicht `IEnumerable<T>`) um neue Überladungen der Methoden `Where`, `Select`, `OrderBy`, `Take`, `Skip`, `GroupBy`, `Any` und `Count`, die als Prädikat einen String erwarten. Dieser String muss eine Syntax aufweisen, die in dem Dokument *Dynamic Expressions.html* beschrieben wird (das Sie im Ordner des Visual-Studio-Beispiels und im Ordner des Beispiels dieses Abschnitts finden).

So können Sie zumindest alle Sequenzen, die `IQueryable` implementieren, dynamisch abfragen. Das ist z. B. für LINQ to SQL der Fall. Sie können aber auch eine Sequenz, die `IEnumerable` oder `IEnumerable<T>` implementiert, über die `AsQueryable` in eine `IQueryable`-Sequenz umwandeln, um dynamische Abfragen auch auf Auflistungen und Arrays ausführen zu können ☺.

Integrieren Sie also die Datei *Dynamic.cs* in Ihr Projekt und fügen Sie eine `using`-Direktive mit dem Namensraum `System.Linq.Dynamic` hinzu. Dann können Sie z. B. die Beispiel-Artikel dynamisch abfragen:


```
// Artikel einlesen
List<Product> products = GetProducts();

// Die Abfragekriterien zusammenstellen
string name = null;
decimal? maxPrice = 17;
int? categoryID = 1;

// Die Abfrage zusammenstellen
string stringQuery = null;
if (String.IsNullOrEmpty(name) == false)
{
   stringQuery = "Name = \"" + name + "\"";
}
if (maxPrice != null)
{
   if (stringQuery != null)
   {
```

Listing 540: Dynamisches Abfragen einer Sequenz

```
      stringQuery += " && ";
   }
   stringQuery += "Price <= " + maxPrice.Value;
}
if (categoryID != null)
{
   if (stringQuery != null)
   {
      stringQuery += " && ";
   }
   stringQuery += "CategoryID == " + categoryID.Value;
}

// Die Liste in eine IQueryable-Sequenz konvertieren
var queryableProducts = products.AsQueryable();

// Die Abfrage ausführen
var query = (stringQuery != null ? queryableProducts.Where(stringQuery) :
   queryableProducts);

// Das Ergebnis auswerten
this.productListView.ItemsSource = query;
```

Listing 540: Dynamisches Abfragen einer Sequenz (Forts.)

Wie Sie im Beispiel sehen, ist die Grundsyntax einer dynamischen Abfrage die, dass Sie sich im String auf Felder oder Eigenschaften beziehen, diese über die Standard-Vergleichsoperatoren (außer `LIKE`) mit Werten vergleichen und mehrere Vergleichsausdrücke über die logischen Operatoren `&&` (oder `and`) und `||` (oder `or`) miteinander verknüpfen. Ein wichtiger Punkt ist auch noch, dass `Where` nicht mit einem leeren String aufgerufen werden kann (weswegen das Beispiel den Abfrage-String daraufhin überprüft).

Hinweis

Das dynamische Abfragen hat aber leider einige Einschränkungen. So können Sie scheinbar nur Sequenzen abfragen, die Objekte mit Feldern oder Eigenschaften verwalten (keine Sequenzen mit einfachen Werten). Außerdem stehen spezielle Operationen wie die `String`-Methode `StartsWith` oder SQL-`LIKE`-ähnliche Operatoren nicht zur Verfügung. In unserem Beispiel bringt das den wesentlichen Nachteil mit sich, dass nicht nach einem Teil des Produktnamens gesucht werden kann. Und Sie sollten beachten, dass eine dynamische Abfrage etwas langsamer ausgeführt wird als eine normale.

285 Ungleichheits-Verknüpfungen

Eine Ungleichheits-Verknüpfung (Non-Equi Join) ermittelt alle Elemente einer Sequenz, die nicht in einer anderen Sequenz (über einen ID-Wert) referenziert werden. Das Prinzip einer Ungleichheits-Verknüpfung ist einfach: Sie fragen die Daten mit zwei `from`-Klauseln ab und setzen für den Vergleich statt des Gleichheits-Operators (==) den Ungleichheits-Operator (!=) ein. Das folgende Beispiel ist allerdings sinnlos:

```
var senselessNonEquiJoin =
   from product in dataContext.Products
   from category in dataContext.Categories
   where product.CategoryID != category.ID
   select product;
```

Sinnvoll werden Ungleichheits-Verknüpfungen, wenn Sie eine normale Verknüpfung mit einer Abfrage auf ungleiche Felder kombinieren. In der Northwind-Datenbank werden z. B. Bestell-Details in der Tabelle *Order Details* verwaltet. *Order Details* referenziert die Tabelle *Products* über die ID der in dieser Tabelle gespeicherten Artikel. In Order Details wird der zum Zeitpunkt des Verkaufs gültige Verkaufspreis der Artikel im Feld *UnitPrice* verwaltet.

Über eine Ungleichheits-Verknüpfung können Sie nun herausfinden, welche Bestelldetails Artikelpreise gespeichert haben, die nicht dem aktuellen Preis entsprechen. Das Ganze erfordert in diesem Fall natürlich LINQ to SQL, einen `DataContext` für die Northwind-Datenbank (im Beispiel `NorthwindDataContext`) und die üblichen Namensraum-Importe.

```
// DataContext erzeugen
NorthwindDataContext dataContext = new NorthwindDataContext();

// Bestelldetails ermitteln, die in den Bestelldetails einen
// Preis aufweisen, der nicht dem aktuellen entspricht
var orderDetailsWithDifferentProductPrice =
   from orderDetail in dataContext.Order_Details
   from product in dataContext.Products
   where orderDetail.ProductID == product.ProductID
      && orderDetail.UnitPrice != product.UnitPrice
   select new
   {
      OrderDetail = orderDetail,
      Product = product
   };
```

Listing 541: Sinnvolle Ungleichheits-Verknüpfung

286 Kreuzprodukt-Verknüpfungen

Eine Kreuzprodukt-Verknüpfung (Cross Join) verknüpft zwei Sequenzen so, dass alle Elemente der einen mit allen Elementen der anderen kombiniert werden. Ein gutes Beispiel für eine solche Verknüpfung ist eine Liste von Vornamen und eine Liste von Nachnamen, wenn Sie alle möglichen Kombinationen der Vor- und Nachnamen erhalten wollen.

Eine Kreuzprodukt-Verknüpfung erhalten Sie in LINQ, indem Sie zwei `from`-Klauseln untereinander schreiben und keine Bedingung angeben:

```
// Je ein Array mit Vor- und Nachnamen erzeugen
string[] firstNames = { "Zaphod", "Ford", "Tricia" };
string[] lastNames = { "Beeblebrox", "Prefect", "McMillan" };

// Die Arrays über einen Cross Join miteinander verknüpfen
var names =
   from firstName in firstNames
   from lastName in lastNames
   select new
      {
         FirstName = firstName,
         LastName = lastName
      };

// Das Ergebnis durchgehen
foreach (var name in names)
{
   Console.WriteLine(name.FirstName + " " + name.LastName);
}
```

Listing 542: Eine Kreuzprodukt-Abfrage

Das Ergebnis dieses Beispiels ist:

```
Zaphod Beeblebrox
Zaphod Prefect
Zaphod McMillan
Ford Beeblebrox
Ford Prefect
Ford McMillan
Tricia Beeblebrox
Tricia Prefect
Tricia McMillan
```

287 Kommaseparierte Dateien (CSV-Dateien) verarbeiten

Das Einlesen von kommaseparierten Dateien (CSV-Dateien) ist eine in der Praxis recht häufig benötigte Technik, die normalerweise rein über einen `StreamReader` gelöst wird. LINQ ermöglicht aber eine objektorientiertere Herangehensweise.

Zur Demonstration der LINQ-Technik zum Einlesen von kommaseparierten Textdateien verwende ich eine Artikel-Textdatei, die folgendermaßen aussieht:

```
ProductId;ProductName;CategoryId;Price
1;Per Anhalter durch die Galaxis;2;9,95
2;Per Anhalter durch die Galaxis (DVD);3;28,99
3;Das C# 2008 Kompendium;1;59,99
4;Das C# 2008 Codebook;1;99,95
5;Die wilde Geschichte vom Wassertrinker;2;12,90
6;2001: Odyssee im Weltraum (DVD);3;9,95
7;2010 - Das Jahr, in dem wir Kontakt aufnehmen (DVD);3;4,79
8;Programmieren lernen;1;24,95
9;Programmieren lernen (E-Book);;15,95
10;Fool on the Hill;2;9,95
```

Der kleine LINQ-Trick basiert auf einer Methode, die die Zeilen der Datei als Auflistung zurückgibt:

```
private static IEnumerable<string> ReadFile(string fileName)
{
   using (StreamReader streamReader = new StreamReader(
      fileName, Encoding.UTF8))
   {
      // Die erste Zeile (die Überschrift) wird ignoriert
      string row = streamReader.ReadLine();

      // Die weiteren Zeilen einlesen und zurückgeben
      while ((row = streamReader.ReadLine()) != null)
      {
         yield return row;
      }
   }
}
```

Listing 543: Methode zum Einlesen einer Textdatei als Auflistung

Über LINQ können Sie diese Methode nun so aufrufen, dass Sie die Zeilen in einzelne Felder auftrennen und das Ergebnis in einen anonymen Typen transformieren, den Sie dann in der Auswertung auslesen:

```
// Den Dateinamen ermitteln
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string fileName = Path.Combine(appPath, "Products.txt");
```

Listing 544: Abfragen und Auswerten der Artikel-Textdatei über LINQ

```
// Die Datei einlesen und über LINQ in eine Sequenz
// von Objekten eines anonymen Typs verarbeiten
var products =
   from row in ReadFile(fileName)
   let fields = row.Split(';')
   select new
   {
      ID = Convert.ToInt32(fields[0]),
      Name = fields[1],
      CategoryID = string.IsNullOrEmpty(
         fields[2]) == false ? (int?)Convert.ToInt32(fields[2]) : null,
      Price = Convert.ToDecimal(fields[3])
   };

// Das Ergebnis auswerten
foreach (var product in products)
{
   Console.WriteLine(product.ID + ": " + product.Name + ": " +
      product.Price);
}
```

Listing 544: Abfragen und Auswerten der Artikel-Textdatei über LINQ (Forts.)

Der wesentliche »Trick« ist, dass die einzelnen Zeilen mit `Split` aufgesplittet werden und das Ergebnis-Array mit `let` in eine interne Variable geschrieben wird. Dieses String-Array wird dann in der Projektion ausgewertet und in einen anonymen Typen transformiert.

Das einzig Schwierige ist dabei das Debuggen, bei dem der Compiler leider die Auswertung der in dem LINQ-Ausdruck verwendeten Variablen nicht zulässt. Das macht die Suche nach der Ursache von Fehlern sehr schwer.

Hinweis

In der Praxis sollten Sie im LINQ-Ausdruck natürlich berücksichtigen, dass eine eingelesene Zeile nicht dem erwarteten Format entspricht. Wie bei XML üblich sollten Sie dann ggf. das gesamte Einlesen mit einer Fehlermeldung abbrechen.

288 Probleme mit der Benennung in LINQ-to-SQL-Modellen lösen

Der LINQ-to-SQL-Designer erzeugt in der deutschen Visual-Studio-Version leider in der Regel falsch benannte Eigenschaften und Klassen. Wenn Sie z. B. für die Tabelle *Products* der englisch benannten Datenbank *Northwind.mdf* ein Datenbankmodell erzeugen, wird die Auflistungs-Eigenschaft der von `DataContext` abgeleiteten Klasse korrekt mit `Products` benannt. Die Daten-Klasse wird aber fälschlicherweise auch in der Mehrzahl benannt. Aber auch mit deutschen Namen hat Visual Studio Probleme: Eine Tabelle mit dem Namen *Kunden* wird in die Klasse `Kunden` (was falsch ist) umgesetzt und in die Eigenschaft `Kunden` (was ok ist).

In der englischen Visual-Studio-Version tritt dieses Problem für *englische* Namen nicht auf. Dort heißen Klasse *Product* und die Eigenschaft *Products*. Das ist sogar unabhängig davon, ob die Tabelle selbst in der Einzahl oder der Mehrzahl benannt ist.

In der ersten Visual-Studio-2008-Version können Sie das Problem im Designer nicht lösen. Wenn Sie den Namen der Datenklasse in die Einzahl ändern, wird auch die Auflistungs-Eigenschaft in der `DataContext`-Klasse im Singular benannt. Theoretisch könnten Sie die vom Designer automatisch erzeugte C#-Datei editieren. Dann würden Ihre Änderungen bei einer späteren Änderung des Modells oder einer Neu-Erzeugung des Codes aber verloren gehen. Sie müssen zur Lösung des Problems die *.dbml*-Datei direkt (über einen Editor) bearbeiten.

Hierfür ändern Sie im Designer zunächst alle Namen, die geändert werden müssen und einstellbar sind. Dazu gehört im Wesentlichen, dass Sie die Namen der Klassen ggf. in die Einzahl ändern und die Namen der Eigenschaften der Beziehungen entsprechend anpassen. Für die Beziehungen passen Sie die Eigenschaften CHILD PROPERTY und PARENT PROPERTY an. CHILD PROPERTY definiert die Eigenschaft, die in der Klasse für die *Mastertabelle* angelegt wird und die die »Kind«-Datensätze (bzw. -Objekte) referenziert. Den Namen dieser Eigenschaft sollten Sie bei einer 1:N-Beziehung in der Mehrzahl angeben. PARENT PROPERTY steht für die Eigenschaft, die in der Klasse für die *Detailtabelle* angelegt wird. Diese sollte immer in der Einzahl angegeben werden (da LINQ to SQL keine N:M-Beziehungen unterstützt).

Speichern Sie die *.dbml*-Datei und öffnen Sie diese dann im Texteditor von Visual Studio. Den entsprechenden Befehl finden Sie im Kontextmenü des Eintrags im Projektmappen-Explorer. Wählen Sie hier ÖFFNEN MIT... und dann QUELLCODE-EDITOR (TEXT).

Ändern Sie in der *.dbml*-Datei nun das Attribut `Member` der `Table`-Elemente auf einen in der Mehrzahl benannten Namen und speichern Sie die Datei.

Listing 545 zeigt dies (auszugsweise) für ein LINQ-to-SQL-Modell, das für die *Categories*- und die *Products*-Tabelle der Northwind-Datenbank erzeugt wurde:


```
<?xml version="1.0" encoding="utf-8"?>
<Database Name="Northwind" Class="NorthwindDataContext" ... />
   <Table Name="dbo.Products" Member="Products">
      <Type Name="Product">
         ...
         <Association Name="Category_Product"
            Member="Categories"
            ThisKey="CategoryID" Type="Category"
            IsForeignKey="true" />
      </Type>
   </Table>

   <Table Name="dbo.Categories" Member="Categories">
      <Type Name="Category">
      ...
```

Listing 545: Auszug aus einer angepassten Datenbank-Modell-Datei für die Northwind-Datenbank

```
      <Association Name="Category_Product" Member="Products"
         OtherKey="CategoryID" Type="Product" />
    </Type>
  </Table>
</Database>
```

Listing 545: Auszug aus einer angepassten Datenbank-Modell-Datei für die Northwind-Datenbank (Forts.)

Ihre Änderungen bleiben auch dann bestehen, wenn Sie das Datenbankmodell nachträglich in Visual Studio ändern.

289 LINQ-to-SQL-Abfragen mit LIKE

Der SQL-Operator `LIKE` ist in der Praxis beim Abfragen von Daten sehr wichtig. LINQ stellt aber keine entsprechende Erweiterungsmethode zur Verfügung. Sie können zwar mit den String-Methoden `StartsWith`, `EndsWith` und `IndexOf` arbeiten, die Flexibilität von `LIKE` erreichen Sie damit aber nicht (oder nur sehr schwer). Falls Sie diesen SQL-Operator nicht kennen: `LIKE` lässt zum Vergleich von Strings die Wildcards `%` und _ zu. `%` steht für beliebig viele beliebige Zeichen. Der Unterstrich steht für genau ein beliebiges Zeichen. Damit können Sie String-Felder sehr flexibel abfragen.

In LINQ to SQL können Sie eine `LIKE`-Abfrage primär für den SQL Server relativ einfach über die Methode `Like` der Klasse `SqlMethods` aus dem Namensraum `System.Data.Linq.SqlClient` ausführen (obwohl die Dokumentation behauptet, diese Methode würde eine `NotSupportedException` auslösen).

So können Sie z. B. die Artikel der Northwind-Datenbank nach Artikeln abfragen, deren Name mit »c« beginnt und mit »g« endet (wobei die Groß- und Kleinschreibung nicht unterschieden wird):

```
using System;
using System.Data.Linq.SqlClient;
using System.Linq;

// DataContext erzeugen
NorthwindDataContext dataContext = new NorthwindDataContext();

// Artikel mit LIKE abfragen
var products =
   from product in dataContext.Products
   where SqlMethods.Like(product.ProductName.ToLower(), "c%g")
   select product;
```

Listing 546: Abfrage auf der Northwind-Datenbank auf einem SQL Server mit LIKE

Hinweis

Für andere Datenbanksysteme funktioniert `Like` nur dann, wenn der entsprechende LINQ-to-SQL-Provider diese Methode unterstützt.

290 SQL direkt ausführen

Wenn Sie in speziellen Fällen in LINQ to SQL eine SQL-Abfrage direkt ausführen wollen, können Sie dies über die `ExecuteQuery`-Methode der `DataContext`-Klasse. Diese Methode erwartet am Typparameter die Klasse, deren Instanzen im Ergebnis zurückgegeben werden sollen. Dabei kann es sich um eine beliebige Klasse handeln, die lediglich einen parameterlosen Standardkonstruktor zur Verfügung stellen muss. Wesentlich ist, dass die Klasse für die in der SQL-Anweisung abgefragten Felder ein gleichnamiges öffentliches, nicht schreibgeschütztes Feld oder eine entsprechende Eigenschaft besitzt. `ExecuteQuery` sucht für alle abgefragten Felder per Reflektion ein öffentliches Feld oder eine öffentliche Eigenschaft, das bzw. die denselben Namen besitzt. Wird ein Feld oder eine Eigenschaft gefunden, schreibt `ExecuteQuery` den Inhalt des Datenbank-Feldes dort hinein.

Am Argument *query* übergeben Sie die Abfrage. In der Abfrage können Sie Platzhalter (`{0}`, `{1}` etc.) einbauen, deren Werte Sie an den folgenden optionalen `Object`-Parametern übergeben.

So können Sie z. B. die Artikel der Northwind-Datenbank in Instanzen einer `ProductSummary`-Klasse abfragen, die zur Kategorie 1 gehören (wofür Sie eigentlich keine Abfrage in SQL-Form benötigen, aber das Beispiel soll möglichst einfach sein):

```
/* Klasse für die abgefragten Artikeldaten */
public class ProductSummary
{
   public int ProductId;
   public string ProductName;
   public decimal? UnitPrice;
}

...

// Eine Instanz des vom LINQ-to-SQL-Modell-Designer generierten
// DataContext erzeugen
NorthwindDataContext dataContext = new NorthwindDataContext();

// Abfragen aller Artikel der Kategorie 1
int categoryId = 1;
var productsOfCategory = dataContext.ExecuteQuery<ProductSummary>(
   @"SELECT * FROM Products WHERE CategoryId = {0}", categoryId);
foreach (var product in productsOfCategory)
```

Listing 547: Abfragen mit einem SQL-String

```
{
   Console.WriteLine(product.ProductName);
}
```

Listing 547: Abfragen mit einem SQL-String (Forts.)

Hinweis

Da `ExecuteQuery` keine Ausnahme wirft, wenn die abgefragten Felder in der Klasse, die am Typparameter übergeben wird, nicht gefunden werden, sollten Sie sehr genau überprüfen, ob die Namen der Felder bzw. Eigenschaften der Klasse mit den abgefragten Datenbankfeldern identisch sind. Für den Fall, dass Sie in dem Typ andere Namen verwenden als in der Datenbank, können Sie die abgefragten Felder in der SQL-Anweisung mit `AS` umbenennen.

291 Die SQL-Anweisung einer LINQ-Abfrage evaluieren

Wenn Sie wissen wollen, welche SQL-Anweisung LINQ to SQL zum Datenbanksystem sendet, können Sie die `Log`-Eigenschaft des `DataContext`-Objekts mit einem `TextWriter` belegen. Sofern Sie an der Konsole programmieren, ist dies einfach, da Sie `Console.Out` in `Log` schreiben können. Besser wäre jedoch, wenn die Ausgabe direkt in das Ausgabefenster von Visual Studio erfolgen würde.

Die Lösung dieses Problems ist die Implementierung einer eigenen `TextWriter`-Klasse, die die geschriebenen Strings in die Visual-Studio-Ausgabe umleitet.

Die folgende Klasse `TraceWriter` habe ich auf der Basis einer Klasse entwickelt, die Kris Vandermotten in einem seiner Blog-Einträge (*www.u2u.info/Blogs/Kris/Lists/Posts/Post.aspx?ID=11*) veröffentlicht hat. Zum Kompilieren müssen Sie die Namensräume `System`, `System.Diagnostics`, `System.Globalization`, `System.IO` und `System.Text` einbinden.

```
public class TraceWriter : TextWriter
{
   /* Gibt an, ob der TraceWriter zerstört wurde */
   private bool isDisposed = false;

   private static UnicodeEncoding encoding;
   /* Die Codierung des TextWriters */
   public override Encoding Encoding
   {
      get
      {
         if (TraceWriter.encoding == null)
         {
            TraceWriter.encoding = new UnicodeEncoding(false, false);
         }
```

Listing 548: Spezieller TextWriter, der die geschriebenen Strings und Zeichen in die Ausgabe von Visual Studio umleitet

```
        return TraceWriter.encoding;
    }
}

/* Gibt die Wichtigkeit der Trace-Nachricht an */
public int Level
{
    private set;
    get;
}

/* Gibt die Kategorie der Trace-Nachricht an */
public string Category
{
    private set;
    get;
}

/* Konstruktor */
public TraceWriter()
    : this(0, Debugger.DefaultCategory)
{
}

/* Konstruktor */
public TraceWriter(int level, string category)
    : this(level, category, CultureInfo.CurrentCulture)
{
}

/* Konstruktor */
public TraceWriter(int level, string category,
    IFormatProvider formatProvider)
    : base(formatProvider)
{
    this.Level = level;
    this.Category = category;
}

/* Gibt das Objekt frei */
protected override void Dispose(bool disposing)
{
    this.isDisposed = true;
    base.Dispose(disposing);
}
```

Listing 548: Spezieller TextWriter, der die geschriebenen Strings und Zeichen in die Ausgabe von Visual Studio umleitet (Forts.)

```
   /* Schreibt ein Zeichen */
   public override void Write(char value)
   {
      if (this.isDisposed)
      {
         throw new ObjectDisposedException(null);
      }
      Debugger.Log(this.Level, this.Category, value.ToString());
   }

   /* Schreibt einen String */
   public override void Write(string value)
   {
      if (this.isDisposed)
      {
         throw new ObjectDisposedException(null);
      }
      if (value != null)
      {
         Debugger.Log(this.Level, this.Category, value);
      }
   }

   /* Schreibt ein Zeichen-Array */
   public override void Write(char[] buffer, int index, int count)
   {
      if (this.isDisposed)
      {
         throw new ObjectDisposedException(null);
      }
      if (buffer == null || index < 0 || count < 0 ||
         buffer.Length - index < count)
      {
         base.Write(buffer, index, count);
      }
      Debugger.Log(this.Level, this.Category, new string(buffer, index,
         count));
   }
}
```

Listing 548: Spezieller TextWriter, der die geschriebenen Strings und Zeichen in die Ausgabe von Visual Studio umleitet (Forts.)

Wenn Sie nun die SQL-Anweisungen, die LINQ to SQL zum Datenbanksystem sendet, im Ausgabefenster von Visual Studio ausgeben wollen, schreiben Sie eine neue Instanz dieser Klasse in die `Log`-Eigenschaft des `DataContext`:

```
NorthwindDataContext dataContext = new NorthwindDataContext();
dataContext.Log = new TraceWriter(0, "LINQ to SQL");
```

Tipp

Eine andere, sehr interessante Möglichkeit, den generierten SQL-String zu evaluieren, ist der *SQL Server Query Visualizer*, den Sie in den Visual-Studio-Beispielen im Ordner *Samples\1031\CSharpSamples\LinqSamples\QueryVisualizer* finden. Diese Beispiele erhalten Sie auch im Internet an der Adresse *code.msdn.microsoft.com/csharpsamples/Release/ProjectReleases.aspx*. Kompilieren Sie das Projekt und kopieren Sie die resultierende *.dll*-Datei in den Ordner *Visual Studio 2008\Visualizers* Ihres Dokumente- bzw. Eigene-Dateien-Ordners. Nach einem Visual-Studio-Neustart sollte der Visualisierer zur Verfügung stehen, wenn Sie im Debugger Ergebnisvariablen von LINQ-to-SQL-Abfragen evaluieren. Der dargestellte SQL-String ist allerdings noch die Rohform der SQL-Anweisung, die schließlich zum Datenbanksystem gesendet wird.

Benutzer und Gruppen

292 Informationen zu den Benutzern eines Computers oder einer Domäne auflisten

Die auf einem Computer oder in einer Domäne registrierten Benutzer können Sie über WMI oder ADSI auslesen. Ich verwende in diesem und den folgenden Rezepten ADSI, da diese Technologie für die Arbeit mit Benutzer- und Gruppenkonten mehr Möglichkeiten bietet als WMI. So ist es z. B. über WMI (nach meinen Erfahrungen und Recherchen in Newsgroups) nicht möglich, Benutzerkonten zu erzeugen oder ein Benutzerkonto einer Gruppe hinzuzufügen, was mit ADSI kein Problem darstellt.

Um alle Benutzer eines Computers oder einer Domäne aufzulisten, können Sie den ADSI-Provider *WinNT* verwenden (der unabhängig vom Namen auch unter anderen Windows-Versionen zur Verfügung steht). Dieser Provider ermöglicht (im Gegensatz zum LDAP-Provider) den direkten Zugriff auf Computer im Netzwerk und bietet daneben (wie der LDAP-Provider) auch den Zugriff auf eine Domäne. Informationen zum WinNT-Provider finden Sie an der Adresse *msdn.microsoft.com/en-us/library/aa772237(VS.85)aspx*.

Die in Listing 550 beschriebene Methode `EnumUsers` setzt diesen Provider ein, um alle Benutzer eines Computers bzw. einer Domäne aufzulisten.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System`, `System.DirectoryServices`, `System.Collections.Generic` und `System.Collections.ObjectModel` importieren. Außerdem müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren. Zur Ausführung ist das Vorhandensein von ADSI auf dem Zielsystem erforderlich.

Da die wichtigsten Eigenschaften eines Benutzers ausgelesen werden sollen, habe ich zunächst eine Klasse `User` deklariert, deren Instanzen einen Benutzer repräsentieren sollen. Die Eigenschaften dieser Klasse sind so deklariert, dass die Werte nur intern (innerhalb der Assembly, in der diese Klasse gespeichert ist) geschrieben werden können. Wird diese Klasse (was natürlich sinnvoll ist) gemeinsam mit der `EnumUsers`-Methode in einer separaten Klassenbibliothek kompiliert, kann ein Programm, das `EnumUsers` verwendet, die Benutzerdaten nur lesen.

```
public class User
{
   /* Der Benutzername */
   public string Name
   {
      get;
      internal set;
   }

   /* Der volle Name des Benutzers */
   public string FullName
```

Listing 549: Klasse zur Verwaltung der wichtigsten Daten eines Benutzers

```
    {
        get;
        internal set;
    }
    /* Die Beschreibung des Benutzers */
    public string Description
    {
        get;
        internal set;
    }

    /* Der maximal verfügbare Speicherplatz (falls Quoting verwendet wird) */
    public int MaxStorage
    {
        get;
        internal set;
    }

    /* Das Home-Verzeichnis des Benutzers */
    public string HomeDirectory
    {
        get;
        internal set;
    }

    /* Datum des letzten Login */
    public DateTime LastLogin
    {
        get;
        internal set;
    }
}
```

Listing 549: Klasse zur Verwaltung der wichtigsten Daten eines Benutzers (Forts.)

Hinweis

Um das Rezept nicht zu komplex zu gestalten habe ich nur die in meinen Augen wichtigsten Eigenschaften eines Benutzers in die Klasse aufgenommen. Im Rezept 297 finden Sie eine Lösung zum Auslesen aller Eigenschaften eines Benutzers.

Die Methode `EnumUsers` fragt alle Benutzer eines Computers oder einer Domäne ab, deren Namen übergeben werden. Diese Methode erzeugt für jeden Benutzer ein `User`-Objekt und fügt dieses an eine lokale Auflistung der generischen `List`-Klasse an. Eine Instanz der ebenfalls generischen Klasse `ReadOnlyCollection` (aus dem Namensraum `System.Collections.ObjectModel`) die mit der `List`-Auflistung initialisiert wurde, wird schließlich zurückgegeben. Damit ist sichergestellt, dass der Aufrufer die Auflistung nicht verändern kann.

`EnumUsers` erzeugt zum Auflisten der Benutzer ein `DirectoryEntry`-Objekt (aus dem Namensraum `System.DirectoryServices`), das als ADSI-Objekt den Computer bzw. die Domäne repräsentiert. Im Konstruktor wird der ADSI-Pfad übergeben, der zuvor je nach dem übergebenen Domänen- und Computernamen für den WinNT-Provider zusammengesetzt wurde. Wird ein Computername und/oder der Name einer Domäne übergeben, repräsentiert der ADSI-Pfad den angegebenen Computer. Wird nur ein Domänenname übergeben, steht der Pfad für die Domäne.

Der Pfad zur Domäne *Galaxy* sieht beispielsweise so aus:

```
WinNT://Galaxy,domain
```

Der Pfad zum Computer *Zaphod* sieht folgendermaßen aus:

```
WinNT://Zaphod,computer
```

Sind Computer und Domäne angegeben, enthält der Pfad beide Angaben:

```
WinNT://Galaxy/Zaphod,computer
```

Auf diese Weise können Sie die lokalen Benutzer eines Computers oder die globalen Benutzer einer Domäne auflisten. Für den Fall, dass an beiden Argumenten `null` übergeben wird, setzt `EnumUsers` den Namen des lokalen Computers in das Argument `machineName` ein.

Da der Zugriff auf Verzeichnisdienste normalerweise geschützt ist, übergibt `EnumUsers` am zweiten und dritten Argument des `DirectoryEntry`-Konstruktors einen Benutzernamen und dessen Passwort. Diesen Benutzer verwendet das `DirectoryEntry`-Objekt zur Authentifizierung gegenüber dem Verzeichnisdienst. Würden Sie diese beiden Argumente weglassen, würde das Objekt das Konto des aktuell in Windows eingeloggten Benutzers verwenden. Da dieses in der Praxis häufig nicht die erforderlichen Rechte besitzt, habe ich `EnumUsers` so entwickelt, dass der für ADSI zu verwendende Benutzer und dessen Passwort übergeben werden können. Sie können aber auch einfach `null` übergeben, wenn das Konto des aktuell eingeloggten Benutzers für die Authentifizierung eingesetzt werden soll.

Danach geht `EnumUsers` über die `Children`-Eigenschaft alle dem ermittelten `DirectoryEntry`-Objekt untergeordneten Objekte durch (die wieder über `DirectoryEntry`-Instanzen repräsentiert werden). Für jedes so ermittelte Objekt wird über dessen Eigenschaft `SchemaClassName` abgefragt, ob es sich um ein Objekt der Schema-Klasse `User` handelt (die ein Bestandteil des WinNT-Providers ist). Ist dies der Fall, erzeugt `EnumUsers` ein neues `User`-Objekt und initialisiert dies zunächst mit dem Namen des Benutzers (der aus der Eigenschaft `Name` des `DirectoryEntry`-Objekts ausgelesen wird). Die anderen Eigenschaften eines Benutzers sind in der generischen `DirectoryEntry`-Klasse nicht direkt bekannt und stehen über die `Properties`-Auflistung zur Verfügung, der Sie den Namen der Eigenschaft übergeben. Da diese Auflistung `object`-Instanzen zurückgibt, müssen diese noch konvertiert werden.

ADSI-Objekte weisen eine Besonderheit im Vergleich zu »normalen« Objekten auf: Ein ADSI-Objekt basiert nicht auf einer Klasse, sondern auf einem so genannten *Schema*. Dieses Schema legt fest, welche Eigenschaften das Objekt besitzen *muss* und welche es besitzen *kann*. Bei der `User`-Klasse sind alle Eigenschaften außer der immer vorhandenen Eigenschaft `Name` optional. Die Abfrage über die `Properties`-Auflistung führt im Falle einer nicht vorhandenen Eigenschaft zur Rückgabe von `null`. `EnumUsers` beachtet dies, indem die Abfrage der Eigenschaften in einer Ausnahmebehandlung erfolgt (die allerdings die Ausnahme verwirft).

```
public static ReadOnlyCollection<User> EnumUsers(string domainName,
   string machineName, string authenticationUser,
   string authenticationPassword)
{
   // Auflistung für das Ergebnis erzeugen
   // und als Basis für die ReadOnlyCollection verwenden
   List<User> userList = new List<User>();
   ReadOnlyCollection<User> resultList =
     new ReadOnlyCollection<User>(userList);
   // Gültigen Rechnernamen für den lokalen Computer erzeugen,
   // falls weder die Domäne noch der Rechnername übergeben wurden
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }

   // ADSI-Pfad für den WinNT-Provider zum Auslesen des Computers bzw.
   // der Domäne zusammenstellen
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }

   // DirectoryEntry-Objekt erzeugen
   DirectoryEntry computerEntry = new DirectoryEntry(
      adsiPath, authenticationUser, authenticationPassword);

   try
   {
      // Alle dem Computer untergeordneten Objekte durchgehen
      foreach (DirectoryEntry de in computerEntry.Children)
      {
         // Überprüfen, ob es sich um ein User-Objekt handelt
         if (de.SchemaClassName.ToLower() == "user")
         {
```

Listing 550: Methode zur Auflistung aller Benutzer eines Computers

```
            // Neues Benutzer-Objekt erzeugen und initialisieren
            User user = new User();
            user.Name = de.Name;
            try
            {
               user.FullName = (string)de.Properties["FullName"].Value;
            }
            catch { }
            try
            {
               user.Description = (string)de.Properties["Description"].Value;
            }
            catch { }
            try
            {
               user.HomeDirectory = (string)de.Properties[
                  "HomeDirectory"].Value;
            }
            catch { }
            try
            {
               user.MaxStorage = (int)de.Properties["MaxStorage"].Value;
            }
            catch { }
            try
            {
               user.LastLogin = (DateTime)de.Properties["LastLogin"].Value;
            }
            catch { }

            // User-Objekt an die Auflistung anfügen
            userList.Add(user);
         }
      }
   }
   finally
   {
      // ADSI-Objekt freigeben
      computerEntry.Dispose();
   }

   // Benutzer-Auflistung zurückgeben
   return resultList;
}
```

Listing 550: Methode zur Auflistung aller Benutzer eines Computers (Forts.)

Die Anwendung der `EnumUsers`-Methode zeigt das folgende Listing, das alle Benutzer des Computers *Zaphod* ermittelt und deren Daten an der Konsole ausgibt:

```
string domainName = null;
string machineName = "Zaphod";
string authenticationUser = null;
string authenticationPassword = null;

// Benutzer abfragen
try
{
   ReadOnlyCollection<User> users =
      EnumUsers(domainName, machineName,
      authenticationUser, authenticationPassword);

   // Benutzer durchgehen und deren Daten ausgeben
   for (int i = 0; i < users.Count; i++)
   {
      Console.WriteLine();
      Console.WriteLine("Name: {0}", users[i].Name);
      Console.WriteLine("Voller Name: {0}", users[i].FullName);
      Console.WriteLine("Beschreibung: {0}", users[i].Description);
      Console.WriteLine("Letzter Login: {0}", users[i].LastLogin);
      Console.WriteLine("Home-Verzeichnis: {0}", users[i].HomeDirectory);
      Console.WriteLine("Maximaler Speicherplatz: {0}", users[i].MaxStorage);
   }
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 551: Anwendung der EnumUsers-Methode

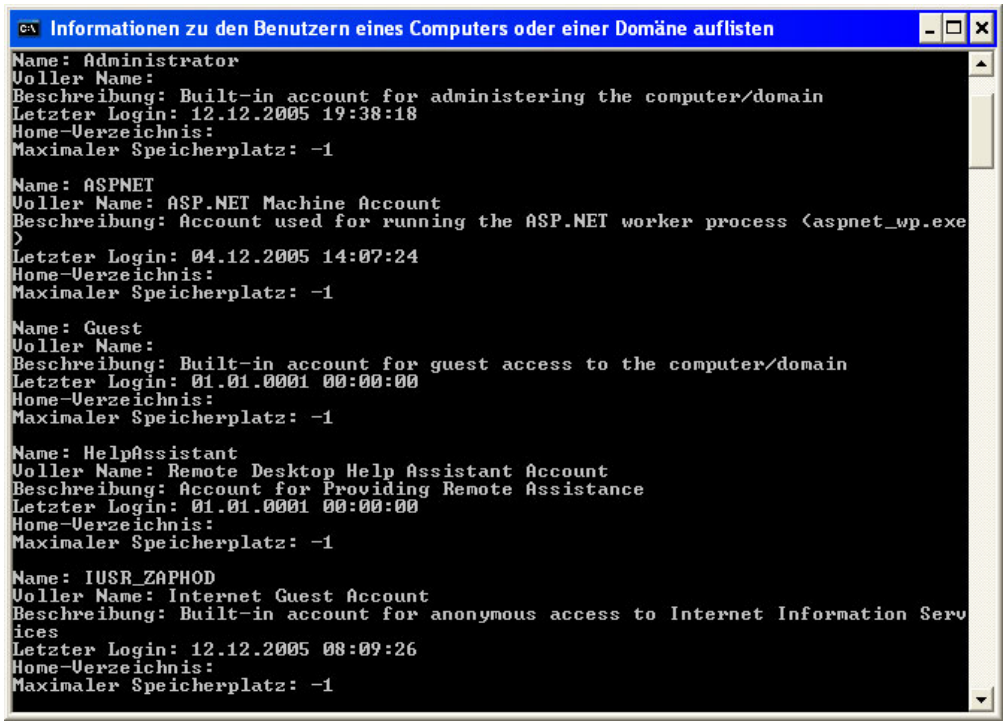


Abbildung 164: Das Beispielprogramm zur Auflistung aller Benutzer eines Computers oder einer Domäne in Aktion

293 Überprüfen, ob ein Benutzerkonto existiert

Die Existenz eines Benutzers können Sie relativ einfach über ADSI überprüfen. Die Methode `UserExists` in Listing 552 überprüft, ob ein Benutzer auf einem Computer oder in einer Domäne existiert. Neben dem Benutzernamen erwartet diese Methode den Namen der Domäne und den des Computers. Um die Existenz eines Benutzers in einer Domäne zu überprüfen, übergeben Sie nur den Domänennamen und am Argument `machineName` `null`. Wenn Sie den Rechner- und/oder Domänennamen angeben, überprüft `UserExists` die Existenz auf einem bestimmten Computer. Für den Fall, dass an den Argumenten `domainName` und `machineName` `null` übergeben wird, schreibt `UserExists` den Namen des lokalen Computers in das Argument `machineName`.

Die Argumente `authenticationUser` und `authenticationPassword` werden wie im Rezept 292 für die Authentifizierung gegenüber ADSI verwendet.

`UserExists` erzeugt zunächst ein `DirectoryEntry`-Objekt, das je nach den übergebenen Domänen- und Rechnernamen entweder den Computer oder die Domäne repräsentiert. Die Existenz des angegebenen Benutzers wird dann über den Aufruf der `Find`-Methode dieses Objekts überprüft. Dieser Methode werden der Name des gesuchten Objekts und dessen ADSI-Klasse übergeben. `Find` sucht im Container, von dem aus diese Methode aufgerufen wurde, und erzeugt eine Ausnahme vom Typ `COMException`, wenn kein passendes Objekt gefunden wurde. Diese Ausnahme nutzt `UserExists` um zu entscheiden, ob der Benutzer existiert.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System` und `System.DirectoryServices` importieren.

```
public static bool UserExists(string userName, string domainName,
   string machineName, string authenticationUser,
   string authenticationPassword)
{
   // Gültigen Rechnernamen für den lokalen Computer erzeugen, falls
   // weder die Domäne noch der Rechnername übergeben wurden
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }

   // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(
      adsiPath, authenticationUser, authenticationPassword);

   try
   {
      DirectoryEntry userEntry = computerEntry.Children.Find(
         userName, "user");
      return true;
   }
   catch
   {
      return false;
   }
}
```

Listing 552: Methode zur Überprüfung der Existenz eines Benutzerkontos

Listing 553 zeigt eine beispielhafte Anwendung der `UserExists`-Methode. Überprüft wird, ob der Benutzer *Administrator* auf dem lokalen Rechner existiert.

```
string domainName = null;
string machineName = null;
string authenticationUser = null;
string authenticationPassword = null;

// Überprüfen, ob der Benutzer Administrator existiert
if (UserExists("Administrator", domainName,
   machineName, authenticationUser, authenticationPassword))
{
   Console.WriteLine("Administrator existiert");
}
else
{
   Console.WriteLine("Administrator existiert nicht");
}
```

Listing 553: Beispielhafte Anwendung der UserExists-Methode

294 Benutzergruppen eines Computers oder einer Domäne auflisten

Um alle auf einem Computer oder in einer Domäne verwalteten Benutzergruppen auszulesen können Sie wie im Rezept 292 ADSI verwenden. Die im Folgenden beschriebene Methode `EnumGroups` basiert auf diesem Rezept. Ich habe den Quellcode lediglich ein wenig verändert. Die Klasse `Group` verwaltet nun die (wenigen) Eigenschaften einer Gruppe. Die Methode `EnumGroups` liest ähnlich der Methode `EnumUsers` nun alle Benutzergruppen ein, speichert deren Eigenschaften in einer `UserGroup`-Instanz, fügt diese einer lokalen `List`-Auflistung hinzu und gibt eine damit initialisierte `ReadOnlyCollection` zurück.

`EnumGroups` werden der Name der Domäne und/oder der Name des Computers übergeben. Wollen Sie die Gruppen einer Domäne abfragen, übergeben Sie lediglich den Domänennamen und tragen am Argument `machineName` `null` ein. Zur Abfrage der lokalen Gruppen eines Computers geben Sie den Computernamen und optional den Namen der Domäne an. Für die Domäne können Sie dann auch `null` übergeben, falls der abgefragte Computer in derselben Domäne verwaltet wird wie der anfragende. Über das Argument `authenticationUser` können Sie einen Benutzer angeben, dessen Konto für die ADSI-Authentifizierung verwendet werden soll. Dessen Passwort geben Sie dann am Argument `authenticationPassword` an. Alternativ können Sie am Argument `authenticationUser` `null` angeben, wenn das Konto des aktuell in Windows eingeloggten Benutzers für die Authentifizierung verwendet werden soll.

Zum Kompilieren des Quellcodes müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System`, `System.DirectoryServices`, `System.Collections.ObjectModel` und `System.Collections.Generic` importieren.

```
/* Verwaltet eine Benutzer-Gruppe */
public class UserGroup
{
   /* Der Name der Gruppe */
   public string Name
   {
      get;
      internal set;
   }

   /* Die Beschreibung der Gruppe */
   public string Description
   {
      get;
      internal set;
   }
}

/* Listet alle Benutzer-Gruppen einer Maschine auf */
public static ReadOnlyCollection<UserGroup> EnumGroups(
   string domainName, string machineName,
   string authenticationUser, string authenticationPassword)
{
   // Basis-Auflistung für die zu ermittelnden Benutzergruppen erzeugen
   List<UserGroup> userGoupList = new List<UserGroup>();

   // Gültigen Rechnernamen für den lokalen Computer erzeugen,
   // falls weder die Domäne noch der Rechnername übergeben wurden
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }

   // DirectoryEntry-Objekt für den Computer oder die Domäne erzeugen
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
```

Listing 554: Methode zum Auflisten der Benutzergruppen eines Computers

```
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   try
   {
      // Alle dem Computer untergeordneten Objekte durchgehen
      foreach (DirectoryEntry de in computerEntry.Children)
      {
         // Überprüfen, ob es sich um ein Group-Objekt handelt
         if (de.SchemaClassName.ToLower() == "group")
         {
            // Neues Group-Objekt erzeugen und initialisieren
            UserGroup group = new UserGroup();
            group.Name = de.Name;
            try
            {
               group.Description = (string)de.Properties["Description"].Value;
            }
            catch { }

            // Group-Objekt an die Auflistung anfügen
            userGoupList.Add(group);
         }
      }
   }
   finally
   {
      // ADSI-Objekt freigeben
      computerEntry.Dispose();
   }

   // Schreibgeschützte Benutzer-Auflistung zurückgeben
   return new ReadOnlyCollection<UserGroup>(userGoupList);
}
```

Listing 554: Methode zum Auflisten der Benutzergruppen eines Computers (Forts.)

Listing 555 zeigt eine Anwendung der `EnumGroups`-Methode, bei der die Daten aller lokalen Gruppen an der Konsole ausgegeben werden.

```
string domainName = null;
string machineName = null;
string bindUser = null;
```

Listing 555: Anwendung der EnumGroups-Methode

```
string authenticationPassword = null;

// Gruppen abfragen
try
{
   ReadOnlyCollection<UserGroup> groups =
      EnumGroups(domainName, machineName,
      bindUser, authenticationPassword);

   // Benutzer durchgehen und deren Daten ausgeben
   for (int i = 0; i < groups.Count; i++)
   {
      Console.WriteLine();
      Console.WriteLine("Name: {0}", groups[i].Name);
      Console.WriteLine("Beschreibung: {0}", groups[i].Description);
   }
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 555: Anwendung der EnumGroups-Methode (Forts.)



Abbildung 165: Das Beispielprogramm listet alle lokalen Gruppen auf

295 Benutzer ermitteln, die einer Gruppe angehören

Wenn Sie abfragen wollen, welche Benutzer einer Gruppe angehören, können Sie dazu ADSI verwenden. Über ein `DirectoryEntry`-Objekt können Sie ein ADSI-Objekt für eine Gruppe erzeugen. Dazu verwenden Sie idealerweise den ADSI-Provider *WinNT*. Das Erzeugen eines Objekts für die Gruppe *Administratoren* auf dem Computer *Zaphod* sieht z. B. so aus:

```
string adsiPath = "WinNT://Zaphod/Administratoren,group";
string authenticationUser = "Administrator";
string authenticationPassword = "bandit";
DirectoryEntry groupEntry = new DirectoryEntry(adsiPath, authenticationUser,
   authenticationPassword);
```

Im Beispiel habe ich wie in den vorhergehenden Rezepten bei der Erzeugung des `DirectoryEntry`-Objekts einen Benutzer (und dessen Passwort) angegeben, der für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll. Würden Sie die letzten zwei Argumente weglassen, würde ADSI den aktuell in Windows eingeloggten Benutzer verwenden (der u. U. nicht über ausreichende Rechte verfügt).

Wollen Sie die Domäne (oder Arbeitsgruppe) des Computers mit einbeziehen (was das Auslesen des Gruppen-Objekts um einiges beschleunigt!), geben Sie diese vor dem Computernamen an:

```
string adsiPath = "WinNT://Galaxy/Zaphod/Administratoren,group";
```

Die vorstehenden Beispiele repräsentieren lokale Gruppen. Zum Auslesen einer globalen (Domänen-)Gruppe geben Sie keinen Computernamen an:

```
string adsiPath = "WinNT://Galaxy/Administratoren,group";
```

Das so erzeugte `DirectoryEntry`-Objekt ist leider nur ein generisches Objekt. Das tatsächlich referenzierte ADSI-Objekt implementiert u. a. die COM-Schnittstellen `IADs` (*msdn.microsoft.com/en-us/library/aa705950(VS.85).aspx*) und `IADsGroup` (*msdn.microsoft.com/en-us/library/aa706021.aspx*), auf deren Methoden Sie über das `DirectoryEntry`-Objekt nicht direkt zugreifen können. Dazu verwenden Sie die Methode `Invoke`, der Sie den Namen der aufzurufenden `IADsGroup`-Methode und ein `object`-Array mit den Argumenten übergeben.

Die Mitglieder einer Gruppe erreichen Sie über die `IADsGroup`-Methode `Members`, die Sie zwar auch über `Invoke` aufrufen können. Diese Methode gibt aber eine Instanz einer Klasse zurück, die die COM-Schnittstelle `IADsMembers` implementiert. Da diese in Ihrem Programm normalerweise nicht bekannt ist, können Sie die Rückgabe nicht ohne weiteres auswerten.

Die Lösung dieses Problems ist recht einfach: Sie referenzieren dazu die COM-Typbibliothek, die die ADSI-Schnittstellen beschreibt. Diese Typbibliothek finden Sie im COM-Register des Verweis-hinzufügen-Dialogs von Visual Studio 2008 unter dem Namen »Active DS Type Library« (Abbildung 166) oder als Datei im Windows-Systemverzeichnis unter dem Namen *activeds.tlb*.

Wenn Sie nun noch den Namensraum `ActiveDs` importieren, kennt Ihr Programm alle ADSI-Schnittstellen und damit auch die Schnittstelle `IADsMembers`. Dann können Sie `Invoke` problemlos ausführen:

```
IADsMembers adsMembers = (IADsMembers)groupEntry.Invoke("Members");
```

Da die Mitglieder eines `IADsGroup`-Objekts auch andere Objekte als `IADsUser`-Instanzen sein können, müssen Sie die Members-Auflistung noch filtern. Danach können Sie diese durchgehen und alle Gruppen-Benutzer auslesen.

Die Methode `EnumGroupMembers` in Listing 556 setzt diese Technik ein. Sie setzt zunächst an Hand der übergebenen Informationen zur Domäne, zum Computer und zur Gruppe den ADSI-Pfad für den WinNT-Provider zusammen und erzeugt mit diesem Pfad ein `DirectoryEntry`-Objekt. Die Argumente `authenticationUser` und `authenticationPassword` werden dabei für die Authentifizierung gegenüber ADSI übergeben. Über die `Invoke`-Methode des `DirectoryEntry`-Objekts wird dann die `Members`-Methode aufgerufen und die Rückgabe in einer Variablen vom Typ `IADsMembers` referenziert. Über die `Filter`-Eigenschaft dieses Objekts reduziert `EnumGroupMembers` die in der Auflistung referenzierten Objekte auf Instanzen der Schema-Klasse `User`. `User` ist die Klasse des WinNT-Providers, die Benutzerinformationen speichert und u. a. die Schnittstelle `IADsUser` implementiert. Danach geht `EnumGroupMembers` die ermittelte Auflistung in einer normalen `foreach`-Schleife durch und hängt die Namen der Benutzer an eine `StringCollection` an, die schließlich zurückgegeben wird.

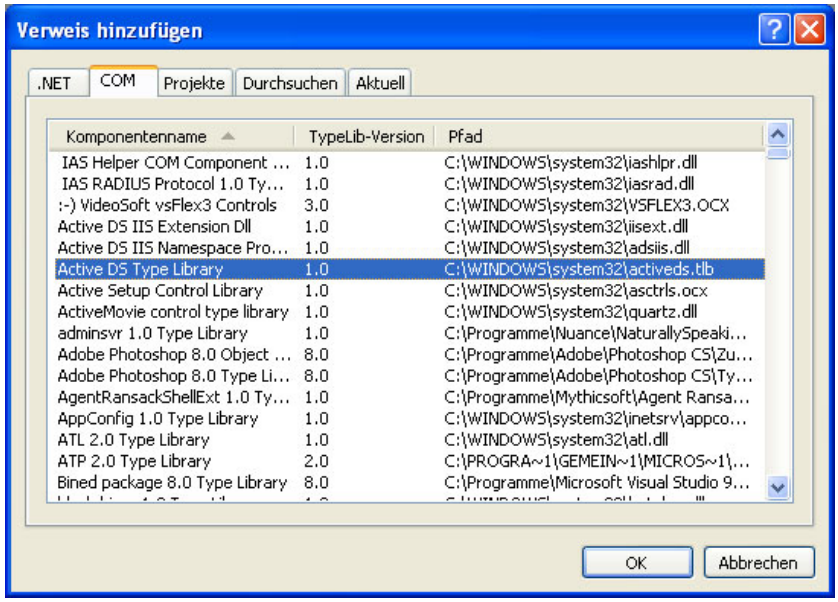


Abbildung 166: Anlegen eines Verweises auf die Active DS Type Library

`EnumGroupMembers` erfordert die Referenzierung der Assembly *System.DirectoryServices.dll* und der ADSI-Bibliothek *activeds.tlb* und den Import der Namensräume `System`, `System.Collections.Generic`, `System.Collections.ObjectModel`, `System.DirectoryServices` und `ActiveDs`.

```
public static ReadOnlyCollection<string> EnumGroupMembers(string domainName,
   string machineName, string groupName,
   string authenticationUser, string authenticationPassword)
{
   // List-Instanz für die ermittelten Benutzernamen erzeugen
   List<string> users = new List<string>();

   // DirectoryEntry-Objekt für die Gruppe holen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/" : "") +
      (machineName != null ? machineName + "/" : "") +
      groupName + ",group";
   DirectoryEntry groupEntry = new DirectoryEntry(
      adsiPath, authenticationUser, authenticationPassword);

   try
   {
      // Member dieser Gruppe einlesen
      IADsMembers adsMembers = (IADsMembers)groupEntry.Invoke("Members");

      // Member so filtern, dass nur User-Objekte übrig bleiben
      adsMembers.Filter = new object[] { "user" };

      // Benutzer durchgehen
      foreach (IADsUser adsUser in adsMembers)
      {
         users.Add(adsUser.Name);
      }
   }
   finally
   {
      // ADSI-Objekt freigeben
      groupEntry.Dispose();
   }

   // Die ermittelten Benutzer zurückgeben
   return new ReadOnlyCollection<string>(users);
}
```

Listing 556: Methode zum Auflisten aller Benutzer einer Gruppe

Ausnahmen, die beim Auslesen der Gruppe entstehen können, werden innerhalb dieser Methode übrigens nicht abgefangen und somit einfach an den Aufrufer weitergereicht. Die Fehlermeldungen, die ADSI liefert, sind in unserem Fall recht aussagekräftig.

Das folgende Listing zeigt eine einfache Anwendung der `EnumGroupMembers`-Methode. Das Programm benötigt den Import der Namensräume `System` und `System.Collections.Specialized`.

```
string domainName = null;
string machineName = "Zaphod";
string groupName = "Administratoren";
string authenticationUser = null;
string authenticationPassword = null;

try
{
   ReadOnlyCollection<string> users = EnumGroupMembers(
     domainName, machineName, groupName, authenticationUser,
     authenticationPassword);
   Console.WriteLine("Benutzer der Gruppe '" + groupName + "':");
   for (int i = 0; i < users.Count; i++)
   {
      Console.WriteLine(users[i]);
   }
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 557: Anwendung der Methode zum Auflisten aller Benutzer einer Gruppe



Abbildung 167: Das Beispielprogramm listet alle Benutzer der Gruppe Administratoren auf

296 Gruppen eines Benutzers abfragen

Wenn Sie ermitteln wollen, welchen Gruppen ein Benutzer angehört, können Sie wie in den vorhergehenden Rezepten ADSI verwenden.

Die Abfrage der Gruppen eines Benutzers ist ähnlich dem vorhergehenden Rezept etwas problematisch. Sie können zwar über den ADSI-Provider *WinNT* ein `User`-Objekt für den Benutzer referenzieren, das die ADSI-Schnittstelle `IADsUser` implementiert. Dessen Gruppen erreichen

Sie aber über die IADsUser-Methode Groups, die ein Objekt zurückgibt, das die Schnittstelle IADsMembers implementiert. Die im generischen DirectoryEntry-Objekt nicht bekannte Groups-Methode können Sie zwar über Invoke aufrufen, die zurückgegebene IADsMembers-Instanz ist aber nicht in Ihrem Programm bekannt.

Als Lösung dieses Problems referenzieren Sie wieder die COM-Typbibliothek *activeds.tlb*, wie ich es im Rezept 295 gezeigt habe. Der Rest entspricht fast ebenso diesem Rezept, nur dass Sie hier statt der Benutzer einer Gruppe die Gruppen eines Benutzers abfragen ☺.

Die im Folgenden beschriebene Methode EnumUserGroups setzt dies um. Übergeben wird, wie schon in den vorhergehenden Rezepten, zunächst ein Domänen- und ein Computername. Geben Sie nur den Domänenname an, sprechen Sie einen globalen (Domänen-)Benutzer an, geben Sie den Computernamen (und optional den Namen der Domäne) an, ermitteln Sie die Gruppen eines lokalen Benutzers. Der Name des Benutzers wird am Argument userName festgelegt. In den letzten Argumenten können Sie Informationen zu dem Benutzerkonto übergeben, das für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll.

EnumUserGroups geht ähnlich EnumGroupMembers vor, nur dass nun ein ADSI-Objekt für den Benutzer erzeugt und dessen Groups-Auflistung durchgegangen wird:

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.DirectoryServices.dll* und die ADSI-Bibliothek *activeds.tlb* referenzieren und die Namensräume System, System.Collections.Generic, System.Collections.ObjectModel, System.DirectoryServices und ActiveDs importieren.

```
public static ReadOnlyCollection<string> EnumUserGroups(string domainName,
   string machineName, string userName, tring authenticationUser,
   string authenticationPassword)
{
   // List-Instanz für die ermittelten Gruppennamen
   List<string> groups = new List<string>();

   // DirectoryEntry-Objekt für den Benutzer erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/" : "") +
      (machineName != null ? machineName + "/" : "") +
      userName + ",user";
   DirectoryEntry userEntry = new DirectoryEntry(
      adsiPath, authenticationUser, authenticationPassword);

   try
   {
```

Listing 558: Methode zum Auflisten aller Gruppen eines Benutzers

```
      // Gruppen dieses Benutzers einlesen
      IADsMembers adsMembers =
         (IADsMembers)userEntry.Invoke("Groups");

      // Member so filtern, dass nur Group-Objekte übrig bleiben
      adsMembers.Filter = new object[] { "group" };

      // Gruppen durchgehen
      foreach (IADsGroup adsGroup in adsMembers)
      {
         groups.Add(adsGroup.Name);
      }
   }
   finally
   {
      // ADSI-Objekt freigeben
      userEntry.Dispose();
   }

   // Die ermittelten Gruppen zurückgeben
   return new ReadOnlyCollection<string>(groups);
}
```

Listing 558: Methode zum Auflisten aller Gruppen eines Benutzers (Forts.)

Listing 559 zeigt eine Anwendung der `EnumUserGroups`-Methode. Das Programm erfordert den Import der Namensräume `System` und `System.Collections.ObjectModel`.

```
string domainName = null;
string machineName = null;
string userName = "Marvin";
string authenticationUser = null;
string authenticationPassword = null;

try
{
   ReadOnlyCollection<string> groups = EnumUserGroups(
      domainName, machineName, userName,
      authenticationUser, authenticationPassword);
   Console.WriteLine("Gruppen des Benutzers '" + userName + "':");
   for (int i = 0; i < groups.Count; i++)
   {
      Console.WriteLine(groups[i]);
   }
}
```

Listing 559: Anwendung der EnumUserGroups-Methode

```
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 559: Anwendung der EnumUserGroups-Methode (Forts.)



Abbildung 168: Auflisten der Gruppen des Benutzers »Marvin«

297 Eigenschaften eines Benutzers auslesen

Benutzerkonten besitzen einige Eigenschaften, die Sie über den Verzeichnisdienst *WinNT* abrufen können, wobei Sie wie in den vorhergehenden Rezepten ADSI verwenden. Die verfügbaren Methoden und Eigenschaften eines damit referenzierten `User`-Objekts beschreibt Microsoft an der Adresse *msdn.microsoft.com/en-us/library/aa746543(VS.85).aspx*.

Nicht allzu einfach ist das Herausfinden der Eigenschaften, die ein `User`-Objekt des WinNT-Providers besitzt. Wie bei ADSI üblich bestimmt eine Schema-Klasse die obligatorischen und optionalen Eigenschaften eines solchen Objekts. Obligatorische Eigenschaften müssen in Objekten implementiert werden, optionale nicht. Ein `User`-Objekt besteht mit Ausnahme der Eigenschaft `Name` ausschließlich aus optionalen Eigenschaften. Beim Herausfinden dieser Eigenschaften hilft die Microsoft-Dokumentation der Schemata des WinNT-Providers (*msdn.microsoft.com/en-us/library/aa746538(VS.85).aspx*).

In reinem ADSI würden Sie über spezielle *Get*- und *Put*-Methoden auf die Eigenschaften eines `User`-Objekts zugreifen. Wenn Sie ein `DirectoryEntry`-Objekt zum Zugriff auf ein solches Objekt verwenden, können Sie die Eigenschaften über dessen `Properties`-Auflistung lesen. Diese Auflistung gibt einfach `null` zurück, wenn Sie den Namen einer Eigenschaft übergeben, die nicht im Objekt existiert.

Die Klasse `User` in Listing 560 implementiert zunächst die möglichen Eigenschaften eines `User`-Objekts. Zum Kompilieren dieser Klasse und des folgenden Programmcodes müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System` und `System.DirectoryServices` importieren.

```
public class User
{
   // Direkte Eigenschaften
   public string Name { get; internal set;}
```

Listing 560: Klasse mit den Eigenschaften für einen Benutzer

```
    public string FullName { get; internal set;}
    public string Description { get; internal set;}
    public int UserFlags { get; internal set;}
    public string Parameters { get; internal set;}
    public int MaxStorage { get; internal set;}
    public string HomeDirectory { get; internal set;}
    public string HomeDirDrive { get; internal set;}
    public string LoginScript { get; internal set;}
    public string Profile { get; internal set;}
    public int MinPasswordLength { get; internal set;}
    public int MinPasswordAge { get; internal set;}
    public int MaxPasswordAge { get; internal set;}
    public int PasswordAge { get; internal set;}
    public bool PasswordExpired { get; internal set;}
    public int BadPasswordAttempts { get; internal set;}
    public int PasswordHistoryLength { get; internal set;}
    public DateTime LastLogin { get; internal set;}
    public DateTime LastLogoff { get; internal set;}
    public int MaxLogins { get; internal set;}
    public string LoginWorkstations { get; internal set;}
    public byte[] LoginHours { get; internal set;}
    public DateTime AccountExpirationDate { get; internal set;}
    public int RasPermissions { get; internal set;}
    public int LockoutObservationInterval { get; internal set;}
    public int PrimaryGroupID { get; internal set;}
    public byte[] objectSid { get; internal set;}
    // Eigenschaften, deren Wert sich aus der Eigenschaft UserFlags
    // berechnet (Eingeschränkte Auswahl!). Siehe
    // msdn.microsoft.com/library/en-us/adsi/adsi/ads_user_flag_enum.asp
    public bool AccountDisabled { get; internal set;}
    public bool AccountLocked { get; internal set;}
    public bool HomeDirRequired { get; internal set;}
    public bool PasswordNotRequired { get; internal set;}
    public bool PasswordCantChange { get; internal set;}
    public bool PasswordNotExpires { get; internal set;}
    public bool EncryptedPasswordAllowed { get; internal set;}
    public bool LocalAccount { get; internal set;}
    public bool NormalAccount { get; internal set;}
    public bool SmartcardRequired { get; internal set;}
}
```

Listing 560: Klasse mit den Eigenschaften für einen Benutzer (Forts.)

Die private (generische) Methode `GetADSIProperty` erleichtert das spätere Auslesen der optionalen Eigenschaften. Diese Methode gibt den Wert einer Eigenschaft des übergebenen `DirectoryEntry`-Objekts zurück, wenn diese gelesen werden konnte. Für den Fall, dass die `Properties`-Auflistung für die abzufragende Eigenschaft `null` speichert, gibt `GetADSIProperty` den Default-(Leer-)Wert des erwarteten Datentyps (der am Typparameter T wird) zurück.

```
private static object GetADSIProperty<T>(DirectoryEntry entry,
   string propertyName)
{
   // Lesen der Eigenschaft
   object propertyValue = entry.Properties[propertyName].Value;

   if (propertyValue != null)
   {
      // Wenn ein Wert gespeichert ist: diesen zurückgeben
      // (boolesche Werte müssen konvertiert werden)
      if (typeof(T) == typeof(bool))
      {
         return (int)propertyValue == 0 ? false : true;
      }
      else
      {
         return propertyValue;
      }
   }
   else
   {
      // Wenn kein Wert gespeichert ist: Den Default-Leerwert des
      // übergebenen Typs zurückgeben
      return default(T);
   }
}
```

Listing 561: Methode zum fehlertoleranten Lesen von ADSI-Eigenschaften

Die Methode `GetUser` liest die Eigenschaften eines Benutzers aus und gibt eine Instanz der Klasse `User` zurück. Ähnlich den Methoden der vorhergehenden Rezepte erwartet diese Methode einen Domänennamen, den Namen des Computers, den Namen des Benutzers und Informationen zu dem Benutzerkonto, das für die Authentifizierung gegenüber dem WinNT-Provider verwendet werden soll. Um globale Benutzerkonten auszulesen, geben Sie nur den Domänennamen und am Argument `machineName` `null` an. Für lokale Benutzer geben Sie den Computernamen und optional auch den Domänennamen an (der ansonsten auch mit `null` belegt werden kann).

`GetUser` erzeugt zunächst ein `DirectoryEntry`-Objekt für den Benutzer und liest dann dessen Eigenschaften über die Methode `GetADSIProperty` ein. Die meisten Eigenschaften können direkt eingelesen werden. Einige Eigenschaften, wie z. B. die Information darüber, ob das Konto deaktiviert ist, werden allerdings im Feld `UserFlags` in Form eines Bitmusters gespeichert. Die dazu verwendeten Konstanten beschreibt Microsoft an der Adresse *msdn.microsoft.com/library/en-us/adsi/adsi/ads_user_flag_enum.asp*. `GetUser` fragt die wichtigsten dieser Flags ab und speichert das Ergebnis in entsprechenden booleschen Eigenschaften der `User`-Klasse.

Das erzeugte und mit den eingelesenen Eigenschaften initialisierte `User`-Objekt wird schließlich zurückgegeben.

```
public static User GetUser(string domainName, string machineName,
   string userName, string authenticationUser, string authenticationPassword)
{
   // Konstanten für die Flags eines Benutzers (Eingeschränkte Auswahl!)
   const int ADS_UF_ACCOUNTDISABLE = 0x0002;
   const int ADS_UF_LOCKOUT = 0x0010;
   const int ADS_UF_HOMEDIR_REQUIRED = 0x0008;
   const int ADS_UF_PASSWD_NOTREQD = 0x0020;
   const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
   const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
   const int ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED = 0x0080;
   const int ADS_UF_NORMAL_ACCOUNT = 0x0200;

   // Gültigen Rechnernamen für den lokalen Computer erzeugen
   // falls weder die Domäne noch der Rechnername übergeben wurden
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }

   // DirectoryEntry-Objekt für den Benutzer erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/" : "") +
      (machineName != null ? machineName + "/" : "") +
      userName + ",user";
   DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   // User-Objekt erzeugen
   User user = new User();

   try
   {
      // Eigenschaften dieses Benutzers einlesen
      user.Name = userName;
      user.Description = (string)GetADSIProperty<string>(userEntry,
         "Description");
      user.FullName = (string)GetADSIProperty<string>(userEntry,
```

Listing 562: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers

```
        "FullName");
    user.UserFlags = (int)GetADSIProperty<int>(userEntry,
        "UserFlags");
    user.Parameters = (string)GetADSIProperty<string>(userEntry,
        "Parameters");
    user.MaxStorage = (int)GetADSIProperty<int>(userEntry,
        "MaxStorage");
    user.HomeDirectory = (string)GetADSIProperty<string>(userEntry,
        "HomeDirectory");
    user.HomeDirDrive = (string)GetADSIProperty<string>(userEntry,
        "HomeDirDrive");
    user.LoginScript = (string)GetADSIProperty<string>(userEntry,
        "LoginScript");
    user.Profile = (string)GetADSIProperty<string>(userEntry,
        "Profile");
    user.MinPasswordLength = (int)GetADSIProperty<int>(userEntry,
        "MinPasswordLength");
    user.MinPasswordAge = (int)GetADSIProperty<int>(userEntry,
        "MinPasswordAge");
    user.MaxPasswordAge = (int)GetADSIProperty<int>(userEntry,
        "MaxPasswordAge");
    user.PasswordAge = (int)GetADSIProperty<int>(userEntry,
        "PasswordAge");
    user.PasswordExpired = (bool)GetADSIProperty<bool>(userEntry,
        "PasswordExpired");
    user.BadPasswordAttempts = (int)GetADSIProperty<int>(userEntry,
        "BadPasswordAttempts");
    user.PasswordHistoryLength = (int)GetADSIProperty<int>(userEntry,
        "PasswordHistoryLength");
    user.LastLogin = (DateTime)GetADSIProperty<DateTime>(userEntry,
        "LastLogin");
    user.LastLogoff = (DateTime)GetADSIProperty<DateTime>(userEntry,
        "LastLogoff");
    user.MaxLogins = (int)GetADSIProperty<int>(userEntry,
        "MaxLogins");
    user.LoginHours = (byte[])GetADSIProperty<byte[]>(userEntry,
        "LoginHours");
    user.RasPermissions = (int)GetADSIProperty<int>(userEntry,
        "RasPermissions");
    user.LockoutObservationInterval = (int)GetADSIProperty<int>(userEntry,
        "LockoutObservationInterval");
    user.PrimaryGroupID = (int)GetADSIProperty<int>(userEntry,
        "PrimaryGroupID");
    user.objectSid = (byte[])GetADSIProperty<byte[]>(userEntry,
        "objectSid");
```

Listing 562: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers (Forts.)

```
      // Auswerten der User-Flags
      if ((user.UserFlags & ADS_UF_ACCOUNTDISABLE) != 0)
      {
         user.AccountDisabled = true;
      }
      if ((user.UserFlags & ADS_UF_DONT_EXPIRE_PASSWD) != 0)
      {
         user.PasswordNotExpires = true;
      }
      if ((user.UserFlags & ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED) != 0)
      {
         user.EncryptedPasswordAllowed = true;
      }
      if ((user.UserFlags & ADS_UF_HOMEDIR_REQUIRED) != 0)
      {
         user.HomeDirRequired = true;
      }
      if ((user.UserFlags & ADS_UF_LOCKOUT) != 0)
      {
         user.AccountLocked = true;
      }
      if ((user.UserFlags & ADS_UF_NORMAL_ACCOUNT) != 0)
      {
         user.NormalAccount = true;
      }
      if ((user.UserFlags & ADS_UF_PASSWD_CANT_CHANGE) != 0)
      {
         user.PasswordCantChange = true;
      }
      if ((user.UserFlags & ADS_UF_PASSWD_NOTREQD) != 0)
      {
         user.PasswordNotRequired = true;
      }
   }
   finally
   {
      // ADSI-Objekt freigeben
      userEntry.Dispose();
   }

   // Das User-Objekt zurückgeben
   return user;
}
```

Listing 562: Methode zum Einlesen der Eigenschaften eines Windows-Benutzers (Forts.)

Listing 563 zeigt eine Anwendung der `GetUser`-Methode. In diesem Beispiel werden nur die wichtigsten Daten an der Konsole ausgegeben.

```
string domainName = null;
string machineName = null;
string userName = "Administrator";
string authenticationUser = null;
string authenticationPassword = null;

// Benutzerdaten auslesen
User user = GetUser(domainName, machineName, userName, authenticationUser,
  authenticationPassword);

// Die wichtigsten Daten ausgeben
Console.WriteLine("Name: {0}",userName);
Console.WriteLine("Description: {0}", user.Description);
Console.WriteLine("FullName: {0}", user.FullName);
Console.WriteLine("AccountDisabled: {0}", user.AccountDisabled);
Console.WriteLine("AccountLocked: {0}", user.AccountLocked);
Console.WriteLine("HomeDirRequired: {0}", user.HomeDirRequired);
Console.WriteLine("PasswordNotRequired: {0}", user. PasswordNotRequired);
Console.WriteLine("PasswordCantChange: {0}", user.PasswordCantChange);
Console.WriteLine("PasswordNotExpires: {0}", user.PasswordNotExpires);
Console.WriteLine("EncryptedPasswordAllowed: {0}",
   user.EncryptedPasswordAllowed);
Console.WriteLine("LocalAccount: {0}", user.LocalAccount);
Console.WriteLine("NormalAccount: {0}", user.NormalAccount);
Console.WriteLine("MaxStorage: {0}", user.MaxStorage);
Console.WriteLine("HomeDirectory: {0}", user.HomeDirectory);
Console.WriteLine("HomeDirDrive: {0}", user.HomeDirDrive);
Console.WriteLine("LoginScript: {0}", user.LoginScript);
Console.WriteLine("Profile: {0}", user.Profile);
Console.WriteLine("PasswordExpired: {0}", user.PasswordExpired);
```

Listing 563: Anwendung der GetUser-Methode

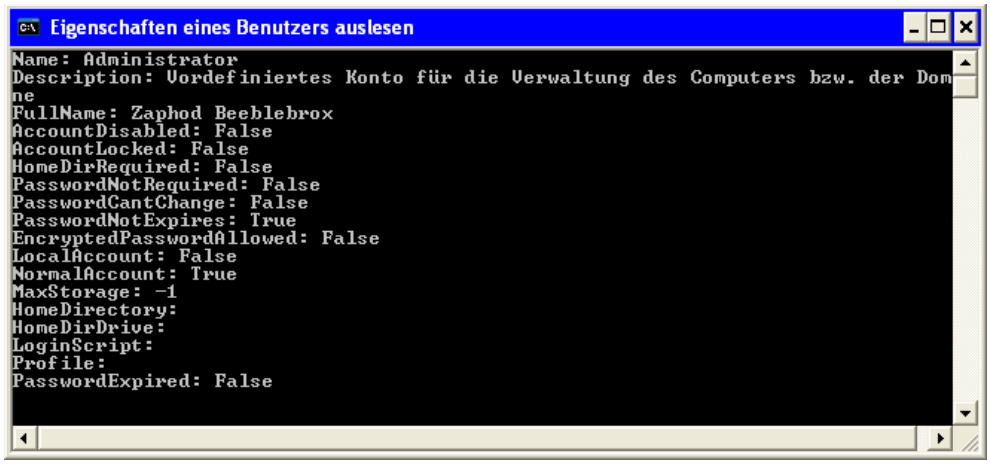


Abbildung 169: Das Beispielprogramm zeigt die wichtigsten Daten des Benutzers »Administrator« an

298 Überprüfen, ob der aktuelle Benutzer einer in Windows vordefinierten Gruppe (z. B. Administratoren) angehört

Zur Überprüfung, ob der Benutzer, unter dessen Konto der aktuelle Thread ausgeführt wird (also in der Regel der Benutzer, der die Anwendung gestartet hat), einer der in Windows vordefinierten Gruppen angehört, können Sie die `IsInRole`-Methode eines `WindowsPrincipal`-Objekts (aus dem Namensraum `System.Security.Principal`) aufrufen, das den aktuellen Benutzer repräsentiert. Dieser Methode können Sie u. a. eine der Eigenschaften der `WindowsBuiltInRole`-Klasse übergeben, die die in Windows vordefinierten Gruppen (Rollen) repräsentieren. `WindowsBuiltInRole.Administrator` steht dabei z. B. für die Gruppe der Administratoren. `IsInRole` gibt `true` zurück, wenn der Benutzer der übergebenen Gruppe angehört.

Das `WindowsPrincipal`-Objekt für den aktuellen Benutzer erhalten Sie über die `CurrentPrincipal`-Eigenschaft der `Thread`-Klasse. Diese Eigenschaft gibt aber lediglich eine Referenz auf die `IPrincipal`-Schnittstelle zurück. Diese besitzt zwar auch eine `IsInRole`-Methode, aber dieser Methode können Sie lediglich einen String übergeben, der die Rolle definiert. Das von `CurrentPrincipal` zurückgegebene Objekt hängt von der aktuellen Principal-Richtline (Principal Policy) ab. Per Voreinstellung gibt `CurrentPrincipal` lediglich ein `GenericPrincipal`-Objekt zurück. Sie müssen die Principal-Richtline zuerst umstellen, sodass Sie ein `WindowsPrincipal`-Objekt erhalten. Und das können Sie über die `SetPrincipalPolicy`-Methode der aktuellen Anwendungsdomäne erreichen. So können Sie zum Beispiel abfragen, ob der aktuelle Benutzer ein Administrator ist:

```
public static bool IsCurrentUserAdministrator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
```

Listing 564: Methode zur Überprüfung, ob der Benutzer, der dem aktuellen Thread zugeordnet ist, Administratorrechte besitzt

```
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.Administrator);
}
```

Listing 564: Methode zur Überprüfung, ob der Benutzer, der dem aktuellen Thread zugeordnet ist, Administratorrechte besitzt (Forts.)

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Security.Principal` und `System.Threading` einbinden.

Zur Vervollständigung habe ich noch Methoden für die restlichen in Windows eingebauten Gruppen implementiert:

```
/* Überprüft, ob der aktuelle Benutzer Benutzerkonten verwalten darf */
public static bool IsCurrentUserAccountOperator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.AccountOperator);
}

/* Überprüft, ob der aktuelle Benutzer Backup-Operator ist */
public static bool IsCurrentUserBackupOperator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.BackupOperator);
}

/* Überprüft, ob der aktuelle Benutzer Gast ist */
public static bool IsCurrentUserGuest()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.Guest);
}

/* Überprüft, ob der aktuelle Benutzer Hauptbenutzer (Power User) ist */
public static bool IsCurrentUserPowerUser()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
```

Listing 565: Methoden zur Überprüfung, ob der aktuelle Benutzer einer bestimmten vordefinierten Gruppe angehört

```
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.PowerUser);
}

/* Überprüft, ob der aktuelle Benutzer Drucker-Operator ist */
public static bool IsCurrentUserPrintOperator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.PrintOperator);
}

/* Überprüft, ob der aktuelle Benutzer Replikator ist */
public static bool IsCurrentUserReplicator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.Replicator);
}

/* Überprüft, ob der aktuelle Benutzer System-Operator ist */
public static bool IsCurrentUserSystemOperator()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.SystemOperator);
}

/* Überprüft, ob der aktuelle Benutzer der Benutzer-Gruppe angehört */
public static bool IsCurrentUserUser()
{
   AppDomain.CurrentDomain.SetPrincipalPolicy(
      PrincipalPolicy.WindowsPrincipal);
   WindowsPrincipal wp = (WindowsPrincipal)Thread.CurrentPrincipal;
   return wp.IsInRole(WindowsBuiltInRole.User);
}
```

Listing 565: Methoden zur Überprüfung, ob der aktuelle Benutzer einer bestimmten vordefinierten Gruppe angehört (Forts.)

299 Gruppen anlegen

Benutzergruppen können Sie relativ problemlos über ADSI anlegen. Dazu erzeugen Sie zunächst ein `DirectoryEntry`-Objekt für den Computer, wobei Sie idealerweise den WinNT-Provider verwenden.

Eine `DirectoryEntry`-Instanz bietet über die `Add`-Methode der `Children`-Auflistung die Möglichkeit, untergeordnete Objekte zu erzeugen. Dazu geben Sie am ersten Argument den Namen des Objekts und am zweiten den Klassennamen an. `Add` gibt eine Referenz auf das erzeugte Objekt zurück. Über dessen `Properties`-Auflistung können Sie dann die Eigenschaften definieren. Da ADSI-Objekte immer im Arbeitsspeicher gecached werden, müssen Sie noch die Methode `CommitChanges` aufrufen um die Änderungen dauerhaft zu schreiben.

Die Methode `CreateGroup` setzt diese Technik ein. Das Argument `groupName` bestimmt den Namen der Gruppe, das Argument `groupDescription` die Beschreibung. Die anderen Argumente dieser Methode entsprechen im Wesentlichen denen der vorhergehenden ADSI-Rezepte.

`CreateGroup` erzeugt zunächst ein `DirectoryEntry`-Objekt für den Computer bzw. die Domäne und ruft dann die `Add`-Methode der `Children`-Auflistung auf um eine neue Gruppe zu erzeugen. Dieser Methode werden der Name des neuen Objekts und der Name der ADSI-Schema-Klasse `Group` übergeben. Über die zurückgegebene Referenz auf das erzeugte Objekt beschreibt `CreateGroup` dann die Eigenschaft `Description`. Die dabei verwendete `Properties`-Auflistung gibt über ihren Indizierer eine Referenz auf eine weitere Auflistung vom Typ `PropertyValueCollection` zurück. Der Grund dafür, dass der Indizierer der `Properties`-Auflistung nicht einen einfachen Wert verwaltet, ist, dass Eigenschaften in ADSI auch aus mehreren Werten bestehen können (ähnlich einem Array). Deshalb muss `CreateGroup` auch die `Add`-Methode der über `Properties` erreichten Werte-Auflistung aufrufen, um die Beschreibung der Gruppe am ersten Wert-Element der Eigenschaft `Description` abzulegen. Die Beachtung dieser Technik ist sehr wichtig, denn der Versuch, direkt in die Eigenschaft `Value` des vom Indizierer der `Properties`-Auflistung zurückgegebenen `PropertyValueCollection`-Objekts zu schreiben, führt zu der wenig aussagekräftigen Ausnahme »Unbekannter Fehler« beim Wegschreiben der geänderten Daten.

Da ADSI alle Objekte zunächst nur im Arbeitsspeicher verwaltet und nicht automatisch in den Verzeichnisdienst zurückschreibt, ruft `CreateGroup` schließlich noch die `CommitChanges`-Methode des `DirectoryEntry`-Objekts auf.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System` und `System.DirectoryServices` importieren.

```
public static void CreateGroup(string domainName, string machineName,
   string groupName, string groupDescription, string authenticationUser,
   string authenticationPassword)
{
   // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
   if (domainName == null && machineName == null)
   {
```

Listing 566: Methode zum Hinzufügen einer Gruppe

```
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(
      adsiPath, authenticationUser, authenticationPassword);

   try
   {
      // Gruppe hinzufügen und Eigenschaften definieren
      DirectoryEntry groupEntry = computerEntry.Children.Add(
         groupName, "group");
      groupEntry.Properties["description"].Add(groupDescription);
      groupEntry.CommitChanges();
   }
   finally
   {
      computerEntry.Dispose();
   }
}
```

Listing 566: Methode zum Hinzufügen einer Gruppe (Forts.)

Sofern bei der Ausführung dieser Methode keine Ausnahme eintritt, wurde die Gruppe erfolgreich erzeugt. Ausnahmen werden nicht abgefangen, da diese recht aussagekräftig sind und einfach an den Aufrufer weitergereicht werden können. Beim Anlegen einer lokalen Gruppe mit einem Namen, der bereits für eine Gruppe oder ein Benutzerkonto verwendet wird, erzeugt `CreateGroup` z. B. eine Ausnahme vom Typ `System.Runtime.InteropServices.COMException` mit der Meldung »Die angegebene lokale Gruppe ist bereits vorhanden« bzw. »Das Konto existiert bereits«.

300 Benutzer anlegen

Das Anlegen eines Benutzerkontos funktioniert ähnlich dem Anlegen einer Gruppe im vorhergehenden Rezept.

Der in Listing 567 beschriebenen Methode `CreateUser` werden neben den bereits in den anderen ADSI-Rezepten verwendeten Argumenten der Name des neuen Benutzers (`userName`), der volle Name (`fullName`), eine Beschreibung (`description`), ein Profilordner (`profile`), ein Login-Skript (`loginScript`), das Home-Verzeichnis (`homeDirectory`), das Passwort (`password`), eine Info darüber, ob das Passwort vom Benutzer geändert werden kann (`cantChangePassword`), und eine Info darüber, ob das Passwort nie abläuft (`passwordDontExpires`), übergeben. Diese Argumente entsprechen im Wesentlichen den Möglichkeiten, die Sie über die Windows-Benutzeradministration haben.

`CreateUser` erzeugt je nach den übergebenen Domänen- und Computernamen zunächst ein `DirectoryEntry`-Objekt für den Computer oder die Domäne. Über die `Add`-Methode der `Children`-Auflistung wird dann ein neues `User`-Objekt erzeugt. Wie bereits beim Erzeugen einer Gruppe werden dazu der Name des neuen Objekts und der Name der Schema-Klasse für dieses Objekt übergeben. Über die Referenz auf das so erzeugte `DirectoryEntry`-Objekt für den neuen Benutzer schreibt `CreateUser` dann die Eigenschaften. Dabei muss wieder wie im vorhergehenden Rezept beachtet werden, dass dazu die `Add`-Methode der vom Indizierer der `Properties`-Auflistung zurückgegebenen `PropertyValueCollection` aufgerufen werden muss.

Die Informationen darüber, ob das Passwort geändert werden kann und ob dieses niemals verfällt, werden in der Eigenschaft `UserFlags` des `User`-Objekts über festgelegte Konstanten verwaltet. `CreateUser` setzt diese Konstanten deswegen je nach den übergebenen Werten in einer `int`-Variablen zusammen und schreibt diese in die Eigenschaft. Da es sich um ein normales Konto handeln soll, wird zusätzlich das Flag `ADS_UF_NORMAL_ACCOUNT` gesetzt. Die möglichen Flags finden Sie an der Adresse *msdn.microsoft.com/library/en-us/adsi/adsi/ads_user_flag_enum.asp*.

Schließlich übergibt `CreateUser` noch das Passwort, was über den Aufruf der nativen Methode `SetPassword` geschieht. Native Methoden werden über die `Invoke`-Methode aufgerufen. `CreateUser` übergibt dieser Methode am ersten Argument den Namen der nativen Methode. Am zweiten Argument nimmt ein `object`-Array die zu übergebenden Argumente auf. In unserem Fall ist das einfach nur das Passwort.

Um die bisher lediglich im Arbeitsspeicher geänderten Daten in den Verzeichnisdienst zu schreiben, ruft `CreateUser` dann noch die Methode `CommitChanges` auf.

`CreateUser` benötigt die Referenzierung der Assembly *System.DirectoryServices.dll* und den Import der Namensräume `System` und `System.DirectoryServices`.

```
public static void CreateUser(string domainName, string machineName,
   string userName, string fullName, string description, string profile,
   string loginScript, string homeDirectory, string password,
   bool cantChangePassword, bool passwordDontExpires,
   string authenticationUser, string authenticationPassword)
```

Listing 567: Methode zum Hinzufügen eines Benutzers

```
{
   // Konstanten für wichtige Flags des User-Objekts
   const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
   const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
   const int ADS_UF_NORMAL_ACCOUNT = 0x0200;

   // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   try
   {
      // Neues User-Objekt erzeugen
      DirectoryEntry userEntry = computerEntry.Children.Add(
         userName, "User");

      // Eigenschaften übergeben
      userEntry.Properties["FullName"].Add(fullName);
      userEntry.Properties["Description"].Add(description);
      userEntry.Properties["Profile"].Add(profile);
      userEntry.Properties["LoginScript"].Add(loginScript);
      userEntry.Properties["HomeDirectory"].Add(homeDirectory);

      // Flags definieren und übergeben
      int flags = ADS_UF_NORMAL_ACCOUNT;
      if (cantChangePassword)
      {
         flags |= ADS_UF_PASSWD_CANT_CHANGE;
```

Listing 567: Methode zum Hinzufügen eines Benutzers (Forts.)

```
      }
      if (passwordDontExpires)
      {
         flags |= ADS_UF_DONT_EXPIRE_PASSWD;
      }
      userEntry.Properties["UserFlags"].Add(flags);

      // Passwort setzen
      userEntry.Invoke("SetPassword", new object[] {password});

      // Geänderte Daten schreiben
      userEntry.CommitChanges();
   }
   finally
   {
      computerEntry.Dispose();
   }
}
```

Listing 567: Methode zum Hinzufügen eines Benutzers (Forts.)

Die beim Erzeugen eines `User`-Objekts möglichen Ausnahmen werden nicht abgefangen, da die Meldungen dieser Ausnahmen im Normalfall sehr aussagekräftig sind und deshalb einfach an den Aufrufer weitergereicht werden können. Wenn Sie z. B. einen Benutzer anlegen wollen, dessen Name bereits für eine Gruppe oder einen Benutzer verwendet wird, erhalten Sie eine Ausnahme vom Typ `System.Runtime.InteropServices.COMException` mit der Meldung »Die angegebene lokale Gruppe ist bereits vorhanden« bzw. »Das Konto existiert bereits«.

301 Eigenschaften eines Benutzers ändern

Wenn Sie die Eigenschaften eines Benutzers ändern wollen, können Sie dazu eine Methode verwenden, die dem Rezept 300 ähnlich ist. Im Unterschied zu dieser wird der Benutzer natürlich nicht erzeugt, sondern über ein `DirectoryEntry`-Objekt referenziert.

Die Methode `ChangeUser` in Listing 568 implementiert dies. Ich verzichte auf eine nähere Beschreibung, da die Arbeitsweise der der Methode `CreateUser` sehr ähnlich ist. Wie bei allen Methoden, die mit ADSI arbeiten, benötigt `ChangeUser` die Referenzierung der Assembly *System.DirectoryServices.dll* und den Import der Namensräume `System` und `System.DirectoryServices`.

```
public static void ChangeUser(string domainName, string machineName,
   string userName, string fullName, string description, string profile,
   string loginScript, string homeDirectory, string password,
   bool cantChangePassword, bool passwordDontExpires,
   string authenticationUser, string authenticationPassword)
```

Listing 568: Methode zum Ändern eines Benutzers

```
{
   // Konstanten für wichtige Flags des User-Objekts
   const int ADS_UF_PASSWD_CANT_CHANGE = 0x0040;
   const int ADS_UF_DONT_EXPIRE_PASSWD = 0x10000;
   const int ADS_UF_NORMAL_ACCOUNT = 0x0200;

   // DirectoryEntry-Objekt für den Benutzer erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/": "") +
      (machineName != null ? machineName + "/": "") +
      userName + ",user";
   DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   try
   {
      // Eigenschaften übergeben
      userEntry.Properties["FullName"].Add(fullName);
      userEntry.Properties["Description"].Add(description);
      userEntry.Properties["Profile"].Add(profile);
      userEntry.Properties["LoginScript"].Add(loginScript);
      userEntry.Properties["HomeDirectory"].Add(homeDirectory);

      // Flags definieren und übergeben
      int flags = ADS_UF_NORMAL_ACCOUNT;
      if (cantChangePassword)
      {
         flags |= ADS_UF_PASSWD_CANT_CHANGE;
      }
      if (passwordDontExpires)
      {
         flags |= ADS_UF_DONT_EXPIRE_PASSWD;
      }
      userEntry.Properties["UserFlags"].Add(flags);

      // Passwort setzen
      userEntry.Invoke("SetPassword", new object[] {password});

      // Geänderte Daten schreiben
      userEntry.CommitChanges();
   }
```

Listing 568: Methode zum Ändern eines Benutzers (Forts.)

```
   finally
   {
      userEntry.Dispose();
   }
}
```

Listing 568: Methode zum Ändern eines Benutzers (Forts.)

302 Benutzer einer Gruppe zuweisen

Über ADSI ist das Zuweisen eines Benutzers zu einer Gruppe recht einfach. Die Methode `AddUserToGroup` in Listing 569 macht genau dies. Die Argumente dieser Methode entsprechen im Wesentlichen denen der vorhergehenden Rezepte. Das Argument `groupName` bezeichnet die Gruppe, das Argument `userName` den Benutzer.

`AddUserToGroup` erzeugt zunächst ein `DirectoryEntry`-Objekt für die Gruppe, wobei der WinNT-Provider eingesetzt wird. Der Pfad zur Gruppe wird je nach den übergebenen Daten mit oder ohne Domäne bzw. Computernamen zusammengesetzt. Auf diese Weise können lokale und globale Gruppen angesprochen werden. Da beim Hinzufügen eines Benutzers dessen ADSI-Pfad benötigt wird, setzt `AddUserToGroup` diesen gleich auch entsprechend zusammen.

Danach ruft `AddUserToGroup` die native Methode `Add` des ADSI-`Group`-Objekts auf um den Benutzer anzufügen. Der Aufruf erfolgt über die `Invoke`-Methode des `DirectoryEntry`-Objekts, das die Gruppe repräsentiert. Dabei werden der Name der nativen Methode und ein `object`-Array mit den Argumenten übergeben. Zum Anfügen eines Benutzers erwartet `Add` die Übergabe eines ADSI-Pfads.

Der Aufruf der `Invoke`-Methode erfolgt in einer Ausnahmebehandlung, die Ausnahmen vom Typ `TargetInvocationException` abfängt. Der Grund dafür ist, dass `Invoke` bei Fehlern, die innerhalb von ADSI auftreten, eine Ausnahme vom Typ `TargetInvocationException` mit der nur wenig aussagekräftigen Fehlermeldung »Ein Aufrufziel hat einen Ausnahmefehler verursacht« erzeugt. Das ist z. B. dann der Fall, wenn ein Benutzer einer Gruppe angefügt werden soll, der dieser bereits angehört. Lediglich die innere Ausnahme, die bei meinen Versuchen vom Typ `COMException` war, enthält die (aussagekräftige) Beschreibung des eigentlichen Fehlers. Die Ausnahmebehandlung überprüft deshalb diese Existenz der inneren Ausnahme und gibt diese einfach weiter. Auf diese Weise kann der Aufrufer aufgetretene Fehler wesentlich einfacher auswerten.

`AddUserToGroup` erfordert die Referenzierung der Assembly *System.DirectoryServices.dll* und den Import der Namensräume `System`, `System.DirectoryServices` und `System.Reflection`.

```
public static void AddUserToGroup(string domainName, string machineName,
   string userName, string groupName, string authenticationUser,
   string authenticationPassword)
{
```

Listing 569: Methode zum Hinzufügen eines Benutzers zu einer Gruppe

```
   // DirectoryEntry-Objekt für die Gruppe holen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string groupPath = "WinNT://" + (
      domainName != null ? domainName + "/" : "") +
      (machineName != null ? machineName + "/" : "") +
      groupName + ",group";
   DirectoryEntry groupEntry = new DirectoryEntry(
      groupPath, authenticationUser, authenticationPassword);

   // Pfad zum Benutzer zusammensetzen
   string userPath = "WinNT://" +
      (domainName != null ? domainName + "/" : "") +
      (machineName != null ? machineName + "/" : "") +
      userName + ",user";

   // Benutzer der Gruppe anfügen
   try
   {
      groupEntry.Invoke("Add", new object[] { userPath });

      // Änderungen wegschreiben
      groupEntry.CommitChanges();
   }
   catch (TargetException ex)
   {
      // Die TargetInvocationException
      // ('Ein Aufrufziel hat einen Ausnahmefehler verursacht')
      // versteckt den eigentlichen Fehler in der inneren
      // Exception, weswegen lediglich diese weitergegeben wird
      if (ex.InnerException != null)
      {
         throw ex.InnerException;
      }
   }
   finally
   {
      groupEntry.Dispose();
   }
}
```

Listing 569: Methode zum Hinzufügen eines Benutzers zu einer Gruppe (Forts.)

303 Benutzer aus einer Gruppe entfernen

Das Entfernen eines Benutzers aus einer Gruppe entspricht fast komplett dem Hinzufügen eines Benutzers, wie ich es im Rezept 302 beschrieben habe. Der einzige Unterschied ist, dass Sie nun anstelle von `Add` die Methode `Remove` des `User`-Objekts aufrufen. Ich verzichte deswegen auf eine nähere Beschreibung der Methode `RemoveUserFromGroup`, die einen Benutzer aus einer Gruppe entfernt.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System`, `System.DirectoryServices` und `System.Reflection` importieren.

```
public static void RemoveUserFromGroup(string domainName,
   string machineName, string userName, string groupName,
   string authenticationUser, string authenticationPassword)
{
   // DirectoryEntry-Objekt für die Gruppe erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string groupPath = "WinNT://" +
      (domainName != null ? domainName + "/": "") +
      (machineName != null ? machineName + "/": "") +
      groupName + ",group";
   DirectoryEntry groupEntry = new DirectoryEntry(groupPath,
      authenticationUser, authenticationPassword);

   // Pfad zum Benutzer zusammensetzen
   string userPath = "WinNT://" +
      (domainName != null ? domainName + "/": "") +
      (machineName != null ? machineName + "/": "") +
      userName + ",user";

   // Benutzer aus der Gruppe entfernen
   try
   {
      groupEntry.Invoke("Remove", new object[] {userPath});
   }
   catch (TargetInvocationException ex)
   {
      // Die TargetInvocationException
      // ('Ein Aufrufziel hat einen Ausnahmefehler verursacht')
      // versteckt den eigentlichen Fehler in der inneren
      // Exception, weswegen lediglich diese weitergegeben wird
      if (ex.InnerException != null)
```

Listing 570: Methode zum Entfernen eines Benutzers aus einer Gruppe

```
      {
         throw ex.InnerException;
      }
      else
      {
         throw ex;
      }
   }

   // Änderungen wegschreiben
   groupEntry.CommitChanges();
}
```

Listing 570: Methode zum Entfernen eines Benutzers aus einer Gruppe (Forts.)

304 Benutzer löschen

Zum Löschen eines Benutzers können Sie (wie bereits schon mehrfach ...) ADSI verwenden.

Die Methode `RemoveUser` löscht einen Benutzer, dessen Name am Argument `userName` übergeben wird. Sie erzeugt dazu abhängig von den Argumenten `domainName` und `computerName` ein `DirectoryEntry`-Objekt für den Computer bzw. die Domäne. Bei der Erzeugung dieses Objekts werden wie in den anderen ADSI-Rezepten Informationen zu dem Benutzerkonto übergeben, das für die Authentifizierung gegenüber dem Verzeichnisdienst verwendet werden soll. Damit kann die Methode auch ausgeführt werden, wenn der aktuell in Windows eingeloggte Benutzer nicht über ausreichende Rechte verfügt.

Über die `Remove`-Methode der `Children`-Auflistung des `DirectoryEntry`-Objekts wird der Benutzer später gelöscht. Da diese Methode ein `DirectoryEntry`-Objekt als Argument erwartet, erzeugt `RemoveUser` dieses zuvor für den angegebenen Benutzer. Ausnahmen werden nicht abgefangen, da diese sehr aussagekräftig sind und einfach an den Aufrufer weitergereicht werden können.

`RemoveUser` benötigt die Referenzierung der Assembly *System.DirectoryServices.dll* und den Import der Namensräume `System`, `System.Reflection` und `System.DirectoryServices`.

```
public static void RemoveUser(string domainName,
   string machineName, string userName,
   string authenticationUser, string authenticationPassword)
{
   // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://";
```

Listing 571: Methode zum Löschen eines Benutzers

```
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   // DirectoryEntry-Objekt für den Benutzer erzeugen
   adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/": "") +
      (machineName != null ? machineName + "/": "") +
      userName + ",user";
   DirectoryEntry userEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   // Benutzer vom Computer bzw. von der Domäne entfernen
   computerEntry.Children.Remove(userEntry);
}
```

Listing 571: Methode zum Löschen eines Benutzers (Forts.)

305 Gruppe löschen

Der Programmcode zum Löschen einer Gruppe entspricht in fast identischer Form dem Löschen eines Benutzers (Rezept 304). Die minimalen Unterschiede sind, dass der Gruppenname übergeben und statt dem `DirectoryEntry`-Objekt für den Benutzer eines für die Gruppe erzeugt wird. Deshalb verzichte ich auch auf eine weitere Beschreibung der Methode `RemoveGroup`.

Zum Kompilieren der `RemoveGroup`-Methode müssen Sie die Assembly *System.DirectoryServices.dll* referenzieren und die Namensräume `System`, `System.Reflection` und `System.DirectoryServices` importieren.

```
public static void RemoveGroup(string domainName,
   string machineName, string groupName,
   string authenticationUser, string authenticationPassword)
{
```

Listing 572: Methode zum Löschen einer Gruppe

```
   // DirectoryEntry-Objekt für den Computer bzw. die Domäne erzeugen
   if (domainName == null && machineName == null)
   {
      machineName = Environment.MachineName;
   }
   string adsiPath = "WinNT://";
   if (domainName != null && machineName != null)
   {
      adsiPath += domainName + "/" + machineName + ",computer";
   }
   else if (machineName != null)
   {
      adsiPath += machineName + ",computer";
   }
   else if (domainName != null)
   {
      adsiPath += domainName + ",domain";
   }
   DirectoryEntry computerEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   // DirectoryEntry-Objekt für die Gruppe erzeugen
   adsiPath = "WinNT://" +
      (domainName != null ? domainName + "/": "") +
      (machineName != null ? machineName + "/": "") +
      groupName + ",group";
   DirectoryEntry groupEntry = new DirectoryEntry(adsiPath,
      authenticationUser, authenticationPassword);

   // Gruppe vom Computer bzw. von der Domäne entfernen
   computerEntry.Children.Remove(groupEntry);
}
```

Listing 572: Methode zum Löschen einer Gruppe (Forts.)

306 Den Namen des aktuellen Benutzers auslesen

Den Namen des aktuellen Benutzers erhalten Sie über die Eigenschaft `UserName` der Klasse `System.Environment`.

```
string currentUserName = System.Environment.UserName;
```

Bei diesem Benutzer handelt es sich um den Benutzer, der dem Thread zugewiesen ist, von dem aus Sie den Programmcode aufrufen. Normalerweise ist das der Benutzer, der in Windows eingeloggt ist. Da Sie ein Programm aber auch im Kontext eines anderen Benutzers ausführen können (siehe Rezept 307), kann es sich auch um einen anderen Benutzer handeln. Eine Lösung, den Namen des in Windows eingeloggten Benutzers auch in solchen Programmen

auszulesen, habe ich leider nicht ermitteln können. Die Windows-API-Funktion `GetUserName`, die sich als Lösung anbietet, liefert leider ebenfalls nur den Namen des Benutzers, der dem aktuellen Thread zugeordnet ist.

307 Ein Programm unter einem spezifischen Benutzerkonto ausführen

Normale Programme werden immer standardmäßig unter dem Konto des aktuell in Windows eingeloggten Benutzers ausgeführt. Lediglich Dienste können über den Dienste-Manager so konfiguriert werden, dass diese unter einem spezifischen Benutzerkonto laufen (normalerweise ist das das Konto *Lokales System*). Ein Programm greift immer mit den Rechten dieses Kontos auf Ressourcen (z. B. auf Dateien oder auf die Registry) zu. In geschützten Systemen, auf denen der Zugriff auf Ressourcen für die einzelnen Benutzer eingeschränkt ist, führt das immer dann zu Problemen, wenn ein Programm eine Ressource bearbeiten muss, für die das aktuell verwendete Benutzerkonto keine ausreichenden Rechte besitzt. So kann es z. B. sein, dass ein Programm Informationen in eine Protokolldatei schreiben muss, die der eingeloggte Benutzer nur lesen darf.

Dieses Problem können Sie lösen, indem Sie den Prozess einer Anwendung mit einem spezifischen Benutzerkonto personifizieren. Dazu rufen Sie die `Impersonate`-Methode eines `WindowsIdentity`-Objekts auf, das einen Windows-Benutzer repräsentiert. Diese Methode gibt eine Referenz auf ein `WindowsImpersonationContext`-Objekt zurück, über dessen `Undo`-Methode Sie die Personifizierung später wieder rückgängig machen können. Dieser Teil der Problemlösung ist einfach.

Dummerweise bietet die `WindowsIdentity`-Klasse auch im .NET Framework 3.5 keine direkte Möglichkeit, eine Instanz für ein spezifisches Benutzerkonto zu erzeugen. An zwei der Konstruktor-Überladungen können Sie zwar einen Benutzernamen im UPN-Format (User-Principal-Name-Format = *Benutzername@Domäne*) übergeben, diese Konstruktoren funktionieren aber für Computer, die Mitglied einer Windows-Server-2003-Domäne sind. An den anderen Konstruktoren können Sie lediglich einen Handle auf ein Windows-Benutzer-Token übergeben. Ein solches Token repräsentiert innerhalb von Windows die Rechte eines bestimmten Benutzers. Ein Benutzer-Token wird üblicherweise mit Windows-API-Funktionen verwendet. Und eine solche rufen Sie dann auch auf, um ein Token für ein Benutzerkonto zu erhalten.

Die Methode `ImpersonateUser` in Listing 573 setzt dies um. Diese Methode kann aufgerufen werden, um eine Anwendung mit einem spezifischen Benutzer zu personifizieren. Sie ruft zunächst die API-Funktion `LogonUser` auf. Diese Funktion erzeugt für den Benutzer, dessen Benutzername, Domäne und Passwort an den ersten Argumenten übergeben werden, ein Benutzer-Token.

Am vierten Argument übergibt `ImpersonateUser` die Konstante `LOGON32_LOGON_NETWORK_CLEARTEXT` (die den Wert 8 besitzt, siehe C++-Headerdatei *winbase.h*). Damit wird erreicht, dass das erzeugte Token die Benutzerinformationen als Klartext zwischenspeichert. Bei Netzwerkzugriffen werden diese Informationen vom entfernten Rechner ausgelesen und für die dortige Authentifizierung verwendet. Nur wenn die Informationen auch vorhanden sind, besitzt ein Token die auf dem entfernten Rechner für das entsprechende Benutzerkonto vergebenen Rechte.

Das Argument dwLogonProvider der LogonUser-Funktion wird mit der Konstante LOGON32_PROVIDER_DEFAULT belegt, damit der voreingestellte Logon-Provider des Systems verwendet wird (alternativ könnten Sie unter Windows 2000 und XP auch den alten NT 4- oder NT 3.5-Provider verwenden). Am letzten Argument wird eine int-Variable übergeben, in die LogonUser den Handle des erzeugten Tokens schreibt.

Über die Auswertung der Rückgabe der Funktion überprüft ImpersonateUser dann, ob der Aufruf erfolgreich war. Im Fehlerfall liest die Methode den letzten API-Fehler aus, generiert über die API-Funktion FormatMessage eine dazu passende Fehlermeldung und wirft damit eine Ausnahme.

War LogonUser erfolgreich, erzeugt ImpersonateUser mit dem erhaltenen Token ein WindowsPrincipal-Objekt und personifiziert die aktuelle Anwendung im Anschluss über dessen Impersonate-Methode. Impersonate gibt eine Referenz auf ein WindowsImpersonationContext-Objekt zurück, über dessen Undo-Methode die Personifizierung rückgängig gemacht werden kann. Damit dies später möglich ist, speichert ImpersonateUser diese Referenz im privaten, statischen Feld winImpersonationContext.

Um das Rückgängigmachen der Personifizierung zu ermöglichen, enthält das Rezept zudem die Methode UndoImpersonation, die einfach nur überprüft, ob das Feld winImpersonationContext ein WindowsImpersonationContext-Objekt referenziert, und die im positiven Fall dessen die Undo-Methode aufruft. Das Programm wird dann wieder unter dem Kontext des Benutzers ausgeführt, unter dem es vor dem Aufruf von ImpersonateUser ausgeführt wurde.

Zum Kompilieren des Quellcodes müssen Sie die Namensräume System, System.Text, System.Runtime.InteropServices und System.Security.Principal importieren.


```
#region Deklaration benötigter API-Funktionen und Konstanten

[DllImport("advapi32.dll", SetLastError = true)]
private static extern int LogonUser(string lpszUsername,
   string lpszDomain, string lpszPassword, int dwLogonType,
   int dwLogonProvider, out int phToken);

[DllImport("kernel32.dll")]
private static extern int FormatMessage(int dwFlags, string lpSource,
   int dwMessageId, int dwLanguageId, StringBuilder lpBuffer, int nSize,
   string[] Arguments);

private const int LOGON32_LOGON_NETWORK_CLEARTEXT = 3;
private const int LOGON32_PROVIDER_DEFAULT = 0;
private const int FORMAT_MESSAGE_FROM_SYSTEM = 0x1000;

#endregion
```

Listing 573: Methoden zur Personifizierung von Anwendungen

```
/* Feld, das ein bei einer Personifizierung erzeugtes
   WindowsImpersonationContext-Objekt referenziert, über das die
   Personifizierung wieder rückgängig gemacht werden kann */
private static WindowsImpersonationContext
   winImpersonationContext = null;

/* Methode zur Personifizierung der aktuellen Anwendung mit einem
   speziellen Benutzerkonto.
   Funktioniert nur unter Windows NT, 2000, XP, Vista, 2003
   und neueren Windows-Versionen */
public static void ImpersonateUser(string domain, string userName,
   string password)
{
   // Benutzer einloggen
   int userToken = 0;
   bool loggedOn = (LogonUser(userName, domain, password,
      LOGON32_LOGON_NETWORK_CLEARTEXT, LOGON32_PROVIDER_DEFAULT,
      out userToken) != 0);

   if (loggedOn == false)
   {
      // Fehler auslesen, in einen Text umwandeln und damit eine
      // Ausnahme werfen
      int apiError = Marshal.GetLastWin32Error();
      StringBuilder errorMessage = new StringBuilder(1024);
      FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, null, apiError,
         0, errorMessage, 1024, null);
      throw new Exception(errorMessage.ToString());
   }

   // WindowsIdentity mit diesem Token erzeugen
   WindowsIdentity identity = new WindowsIdentity((IntPtr)userToken);

   // Prozess mit der ermittelten Windows-Identität personifizieren
   winImpersonationContext = identity.Impersonate();
}

/* Macht eine zuvor über ImpersonateUser ausgeführte Impersonierung
   rückgängig */
public static void UndoImpersonation()
{
   if (winImpersonationContext != null)
   {
      winImpersonationContext.Undo();
   }
}
```

Listing 573: Methoden zur Personifizierung von Anwendungen (Forts.)

Voraussetzung für die erfolgreiche Ausführung dieser Methoden ist:

- dass die Methoden unter Windows NT 4, Windows 2000, Windows XP, Vista oder einer neueren Windows-Version ausgeführt werden,
- dass der Prozess, von dem aus die Methoden aufgerufen werden, unter Windows 2000 (und wahrscheinlich auch unter NT, nicht aber unter XP und Vista) das Privileg SE_TCB_NAME (siehe unten) besitzt
- und dass das angegebene Benutzerkonto das Recht besitzt, sich interaktiv anzumelden.

Unter Windows 2000 ist es leider erforderlich, dass der Benutzer, unter dessen Konto das Programm ausgeführt wird, das Privileg `SE_TCB_NAME` besitzt. Dieses Privileg vergeben Sie über die lokalen Sicherheitseinstellungen (SYSTEMSTEUERUNG / VERWALTUNG / LOKALE SICHERHEITSRICHTLINIE / SICHERHEITSEINSTELLUNGEN / LOKALE RICHTLINIEN / ZUWEISEN VON BENUTZERRECHTEN). Dort wird dieses Privileg als EINSETZEN ALS TEIL DES BETRIEBSSYSTEMS bezeichnet (was eine etwas unklare Übersetzung für das originale »Act as part of the operating system« ist). Laut der Microsoft-Windows-2000-Dokumentation erlaubt dieses Privileg einem Prozess, sich als jeder Benutzer zu authentifizieren (was ja unser Ziel ist). Der Benutzer erhält damit scheinbar keine weiteren spezifischen Rechte (was in gesicherten Systemen fatal wäre). Informationen zu diesem Privileg finden Sie, wenn Sie bei *msdn.microsoft.com* nach »Act as part of the operating system« suchen (die URL der entsprechenden Seite sieht so aus, als würde sie nicht allzu lange bestehen bleiben ...).

Zum Testen der Methoden erzeugen Sie z. B. eine Textdatei (*hitchhiker.txt*) und entfernen die Leserechte auf dieser Datei für alle Benutzer. Vergeben Sie volle Rechte für einen bestimmten Benutzer (im Beispiel ist das der Benutzer *Ford*) und führen Sie das folgende Programm aus, wobei Sie die Variable `userName` mit dem Benutzernamen belegen und die Variable `password` mit dem Passwort des Benutzers. Die Textdatei *hitchhiker.txt* wird beim Einlesen in dem Ordner erwartet, in dem das Programm gespeichert ist.

Das Programm erfordert die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import der Namensräume `System`, `System.IO`, `System.Text`, `System.Windows.Forms` und `System.Security.Principal`.

```
// Benutzerdaten für die Personifizierung
string userName = "Ford";
string password = "Handtuch";

try
{
   // Den Namen des aktuellen Benutzers ausgeben
   WindowsIdentity identity = WindowsIdentity.GetCurrent();
   Console.WriteLine("Aktueller Benutzer: {0}", identity.Name);

   // Aktion ausführen, zu der das interaktive Konto keine Rechte besitzt
   // Anmerkung: Sie sollten die Datei Demo.txt in den Programmordner
```

Listing 574: Beispielhafte Anwendung der zur Personifizierung von Anwendungen

```
   // kopieren und allen Benutzern außer Ford die Leserechte entziehen
   StreamReader sr = new StreamReader(Path.Combine(Application.StartupPath,
      "hitchhiker.txt"), Encoding.Default);
   string row = sr.ReadLine();
   Console.WriteLine(row);
   sr.Close();
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}

Console.WriteLine();

try
{
   // Programm personifizieren
   ImpersonateUser(null, userName, password);

   // Den Namen des nun aktuellen Benutzers ausgeben
   WindowsIdentity identity = WindowsIdentity.GetCurrent();
   Console.WriteLine("Aktueller Benutzer: {0}", identity.Name);

   // Aktion ausführen, zu der das neue Konto nun (hoffentlich) Rechte
   // besitzt
   StreamReader sr = new StreamReader(Path.Combine(
      Application.StartupPath, "hitchhiker.txt"), Encoding.Default);
   string row = sr.ReadLine();
   Console.WriteLine(row);
   sr.Close();
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}

// Personifizierung rückgängig machen
UndoImpersonation();
```

Listing 574: Beispielhafte Anwendung der zur Personifizierung von Anwendungen (Forts.)

Die Methoden zur Personifizierung funktionierten in meinen Tests einwandfrei. Ein kleines Problem bleibt aber noch bestehen: `LogonUser` erzeugte bei meinen Versuchen unter Windows XP dummerweise auch dann ein Token, wenn ein nicht vorhandener Benutzername angegeben wurde. Dieses Token repräsentiert dann den Benutzer *Gast* und besitzt natürlich die sehr eingeschränkten Rechte dieses Benutzers.

Problematisch wird dies, wenn eine Anwendung `ImpersonateUser` einen nicht (mehr) existierenden Benutzernamen übergibt und der Zugriff auf Ressourcen dann zu Ausnahmen führt. Die Ursache dieses Fehlers zu finden kann in dem Fall u. U. recht problematisch sein.

Unter Windows 2000 trat dieses Problem übrigens nicht auf.

Um zu vermeiden, dass ein Programm fehlerhaft ausgeführt wird, wenn das angegebene Benutzerkonto nicht existiert, könnte `ImpersonateUser` das erzeugte `WindowsPrincipal`-Objekt daraufhin überprüfen, ob es die Benutzerinformationen speichert, die übergeben wurden. Im negativen Fall könnte die Methode dann eine Ausnahme generieren.

Dabei entstehen aber zu viele Unwägbarkeiten. Die Eigenschaft `Name` eines `WindowsPrincipal`-Objekts verwaltet den Namen in der Form »*Domäne**Benutzername*«. Übergeben werden können (zurzeit) aber ein normaler Benutzername mit oder ohne Domäne und ein Benutzername im UPN-Format (*Benutzer@DNS-Domäne*). Der Vergleich der übergebenen Benutzerinformationen mit den Informationen im Objekt wäre sehr schwierig und zudem recht unsicher.

Die andere Idee, eine Überprüfung auf den Benutzer *Gast*, ist leider auch nicht allzu sicher. Die Eigenschaft `IsGuest` eines `WindowsPrincipal`-Objekts gibt bedauerlicherweise bei den automatisch erzeugten Gast-Token `false` zurück. Und der Vergleich auf den Namen *Gast* würde das Programm auf Deutschland einschränken, da der Gast-Benutzer in anderen Ländern natürlich anders benannt ist.

Eine Lösung des Problems wäre, vor dem Erzeugen des Tokens zu überprüfen, ob der angegebene Benutzer überhaupt existiert. Dazu können Sie die Methode `UserExists` aus dem Rezept 293 verwenden. Und dann müsste eigentlich alles funktionieren ☺.

Sicherheit

308 Strings sicher im Programm verwalten

Viele Programme verwalten geheime Daten wie Passwörter und Kreditkarteninformationen in String-Instanzen. Ein großes Problem dieser Verwaltung ist, dass normale Strings im Programm nicht verschlüsselt verwaltet werden. Zudem sind Strings im Speicher nicht an eine bestimmte Position gebunden, so dass die CLR diese wahlfrei verschieben kann und damit mehrere Kopien erzeugt (bis der Garbage Collector diese aus dem Speicher entfernt). Hinzu kommt, dass bei jeder Änderung eines Strings eine neue Kopie erzeugt wird und der alte String im Speicher verbleibt. Ein Angreifer, der Zugriff auf den Speicherbereich der Anwendung besitzt, kann diese Strings relativ einfach lesen.

Um dies zu verhindern können Sie geheime Texte in einer Instanz der Klasse `SecureString` aus dem Namensraum `System.Security` verwalten. `SecureString` speichert einen String verschlüsselt, wobei das »Data Protection API« (DPAPI) verwendet wird. DPAPI ist ein API des Betriebssystems und steht erst ab Windows 2000 Service Pack 3 zur Verfügung. Für die älteren Windows-Versionen 98, Me und NT können Sie `SecureString` also nicht verwenden.

Neben der Verschlüsselung wird der verschlüsselte String im Speicher festgelegt (gepinnt), so dass die CLR diesen nicht beliebig verschieben kann. Dies erschwert einem Angreifer zusätzlich das Auslesen von geheimen Daten.

Die einzigen (und schwierigen) Probleme sind, die Daten in die `SecureString`-Instanz zu speichern und daraus auszulesen. `SecureString` bietet aus Sicherheitsgründen keine Möglichkeit einen normalen String zuzuweisen oder einzufügen. Das Problem dabei wäre, dass der normale String wieder lesbar wäre und gegebenenfalls für eine längere Zeit im Speicher verbleiben würde. Deswegen können Sie lediglich im Konstruktor einen Zeiger (!) auf ein `char`-Array übergeben oder über die `AppendChar`-Methode einzelne Zeichen anfügen. Das erschwert natürlich das Handling enorm. Besonders schwierig wird die Eingabe sichererer Daten durch den Benutzer. Wenn Sie dazu eine `Windows.Forms-TextBox` verwenden, den eingegebenen String aus der `Text`-Eigenschaft auslesen und Zeichen für Zeichen der `SecureString`-Instanz hinzufügen, haben Sie nicht allzu viel erreicht. Die `Text`-Eigenschaft der `TextBox` referenziert ja einen normalen String, der wie jeder andere String gegebenenfalls für längere Zeit im Speicher verbleibt und für einen Angreifer lesbar ist.

Hinweis

Das WPF-Steuerelement `PasswordBox` verwendet `SecureString` bereits zur Verwaltung des vom Benutzer oder vom Programm gesetzten Passworts: Die `String`-Eigenschaft `Password` verwaltet ihren Wert über einen (internen) `TextContainer`, dessen `SetPassword`-Methode einen `SecureString` übergeben bekommt. Aufgrund der mageren Dokumentation der `PasswordBox`-Klasse ist es schwer zu sagen, ob dieses Steuerelement in allen Bereichen (z. B. beim Einfügen von Text über die Zwischenablage) wirklich sicher ist. Und nur der Gerechtigkeit halber: Meine `SecureStringTextBox` war eher da ☺ (siehe mein C# Codebook 2.0).

Eine Lösung des String-Sicherheitsproblems in einer Windows.Forms-TextBox wäre, jedes einzelne eingegebene Zeichen abzufangen, hinzuzufügen und der TextBox z. B. als Stern (*) umgewandelt zu übergeben. Dabei müssen Sie jedoch auch die Rück-Taste berücksichtigen und bei der Eingabe gegebenenfalls davon ausgehen, dass die Cursorposition über die Cursor-Tasten oder die Maus geändert oder dass ein Teil des Textes markiert wurde.

Das Beispiel in Listing 575 zeigt einen Ansatz der Lösung des Eingabeproblems. Die in diesem Beispiel implementierte SecureStringTextBox liefert nur eine Teil-Lösung, nämlich das Abfangen der normalen Tasten und der Rück-Taste. Die Klasse setzt zunächst im Konstruktor das Default-Kontextmenü zurück um dessen Bearbeitungsmöglichkeiten abzuschalten. Da die Pos1- und die Cursor-Tasten nicht unterstützt werden, fängt SecureStringTextBox diese in der überschriebenen OnKeyDown-Methode ab und gibt das Ereignis als behandelt an, damit die Betätigung der Tastatur nicht weitergegeben wird. In der OnKeyPress-Methode werden die Tastenbetätigungen abgefangen und in die SecureString-Instanz geschrieben. In OnMouseDown löscht die TextBox die gesamte Eingabe um eine Verschiebung des Eingabecursors durch die Maus zu verhindern. Schließlich überschreibt SecureStringTextBox noch die WndProc-Methode um das Einfügen über die Zwischenablage zu verhindern. Ich weiß: Das ist nur eine Notlösung, aber die Implementierung einer vollständig funktionierenden sicheren TextBox ist an dieser Stelle einfach zu aufwändig. Außerdem bietet WPF ja bereits die (hoffentlich) sichere PasswordBox.

SecureStringTextBox erfordert den Import der Namensräume System, System.Windows.Forms und System.Security.

```
public sealed class SecureStringTextBox : TextBox
{
   /* Verwaltet das sichere Passwort */
   public SecureString SecureInput = new SecureString();

   /* Konstruktor */
   public SecureStringTextBox()
   {
      // Das Kontextmenü entfernen um ein Einfügen über die
      // Zwischenablage zu verhindern
      this.ContextMenu = new ContextMenu();

      // Password-Zeichen vordefinieren
      this.PasswordChar = '*';
   }

   /* Verwirft die Lösch- und die Cursor-Tasten */
   protected override void OnKeyDown(KeyEventArgs e)
   {
      switch (e.KeyCode)
      {
```

Listing 575: Eine einfache TextBox für sichere Eingaben

```
            // Die Pos1-, die Lösch- und die Cursortasten werden
            // nicht unterstützt und deswegen einfach ignoriert
            case Keys.Home:
            case Keys.Left:
            case Keys.Right:
            case Keys.Up:
            case Keys.Down:
               e.Handled = true;
               break;
         }
      }

      /* Speichert jedes eingegebene Zeichen in den sicheren String
         und gibt stattdessen einen Stern aus */
      protected override void OnKeyPress(KeyPressEventArgs e)
      {
         if (e.KeyChar == (char)Keys.Back)
         {
            // Das letzte Zeichen in der sicheren Eingabe löschen
            if (this.SecureInput.Length > 0)
            {
               this.SecureInput.RemoveAt(this.SecureInput.Length - 1);
            }
         }
         else
         {
            // Abfragen, ob es sich um eine Taste mit einem Wert
            // größer 31 handelt (bei STRG-V wird die Taste 22 übergeben!?)
            if (e.KeyChar > (char)31)
            {
               // Das eingegebene Zeichen hinzufügen ...
               this.SecureInput.AppendChar(e.KeyChar);

               // und in das Passwortzeichen umwandeln
               e.KeyChar = this.PasswordChar;
            }
         }
      }

      /* Löscht die sichere Eingabe, wenn die Maus
         auf dem Steuerelement betätigt wird */
      protected override void OnMouseDown(MouseEventArgs e)
      {
         // Passwort löschen
         this.SecureInput.Clear();
```

Listing 575: Eine einfache TextBox für sichere Eingaben (Forts.)

```
      this.Clear();
   }

   /* Fängt das Einfügen über die Zwischenablage ab */
   protected override void WndProc(ref Message m)
   {
      const int WM_PASTE = 0x0302;
      switch (m.Msg)
      {
         case WM_PASTE:
            // Nichts machen
            break;

         default:
            // Die geerbte Methode aufrufen um die Nachricht weiter
            // zu verarbeiten
            base.WndProc(ref m);
            break;
      }
   }
}
```

Listing 575: Eine einfache TextBox für sichere Eingaben (Forts.)

Die `SecureStringTextBox` können Sie nun auf einem Formular einsetzen. Über die Eigenschaft `SecureInput` erreichen Sie den sicheren String. Nach der Eingabe können Sie diesen über die `MakeReadOnly`-Methode schreibschützen. Damit kann dieser nicht mehr verändert werden.

Und dann beginnen die nächsten Probleme. Die meisten .NET-Klassen, die Daten verwalten, die eigentlich geheim gehalten werden müssen (wie z. B. die `NetworkCredential`-Klasse in ihrer `Password`-Eigenschaft), verwalten diese Daten immer noch in normalen Strings und bieten keine äquivalente `SecureString`-Eigenschaft. Lediglich die Klassen `System.Diagnostics.ProcessStartInfo` (`Password`-Eigenschaft), `System.Diagnostics.Eventing.Reader.EventLogSession` (Konstruktor), `System.Security.Cryptography.X509Certificates.X509Certificate2` (`Import`-Methode), `System.Security.Cryptography.CspParameters` (`KeyPassword`-Eigenschaft) und `System.Windows.Controls.PasswordBox` (die WPF-Passwort-`TextBox`, über die `String`-Eigenschaft `Password`) arbeiten nach meinen Recherchen bereits mit sicheren Strings.

Um einen sicheren String an Instanzen der Klassen übergeben zu können, die noch nicht mit sicheren Strings arbeiten, müssen Sie den sicheren String in einen normalen String umwandeln. Das ist jedoch nicht direkt möglich, denn Microsoft hat `SecureString` so entwickelt, dass Instanzen dieser Klasse nicht direkt in einen String umgewandelt oder mit einem solchen verglichen werden können. Die Begründung dafür ist, dass der Verzicht auf diese Möglichkeiten die Sicherheit dieser Klasse wesentlich erhöht. Bei der Bearbeitung einer `SecureString`-Instanz helfen aber einige Methoden der `Marshal`-Klasse (aus dem Namensraum `System.Runtime.InteropServices`). Über `SecureStringToBSTR` können Sie einen sicheren String zum Beispiel in einen Basic-String (BSTR) umwandeln. Sie erhalten dann lediglich einen `IntPtr`-Wert zurück, der die Adresse des Strings (im nicht verwalteten Speicher) beinhaltet. Ein Basic-String (der eigentlich

aus der COM-Welt stammt) unterliegt nicht der CLR und kann nach der Verwendung über `Marshal.ZeroFreeBSTR` mit 0-Zeichen gefüllt werden um die Sicherheit wieder zu erhöhen. Dummerweise kann man mit einem solchen String(-Zeiger) in einem .NET-Program nicht allzu viel anfangen. API-Funktionen, die meist mit Basic-Strings arbeiten, können Sie diesen `IntPtr` wohl direkt übergeben. Für .NET-Klassen, die mit .NET-Unicode-Strings arbeiten, müssen Sie den Basic-String aber noch in einen normalen String umwandeln. Dabei hilft die Methode `PtrToStringUni`, der Sie diese Adresse des Basic-Strings übergeben:

```
// Den sicheren String in einen Basic-String umwandeln
IntPtr bstrPointer = Marshal.SecureStringToBSTR(secureString);

// Mit dem Basic-String arbeiten. Hier:
// Den Basic-String in einen normalen String umwandeln und auswerten
string password = Marshal.PtrToStringUni(bstrPointer);

...

// Den Basic-String mit 0-Zeichen füllen um die Sicherheit zu erhöhen
Marshal.ZeroFreeBSTR(bstrPointer);
```

Listing 576: Umwandeln eines SecureString in einen normalen String

Dabei müssen Sie natürlich beachten, dass Sie jetzt wieder mit normalen Strings arbeiten und diese aus dem Speicher des Programms auslesbar sind. Der Zeitraum, in dem die unsicheren Strings im Programm verwaltet werden, wird aber auf diese Weise um einiges reduziert.

309 Daten symmetrisch ver- und entschlüsseln

Häufig müssen geheime Daten in Konfigurationsdateien oder Datenbanken gespeichert, über das Internet versendet werden oder Ähnliches. Diese Daten sollten Sie normalerweise nicht unverschlüsselt speichern bzw. übertragen.

Das .NET Framework bietet mit den Klassen des Namensraums `System.Security.Cryptography` eine Vielzahl an Möglichkeiten, Daten zu verschlüsseln. Neben den wichtigen Verschlüsselungs-Algorithmen für eine symmetrische und asymmetrische Verschlüsselung werden auch Hashing-Verfahren, digitale Zertifikate und XML-Signaturen unterstützt.

In diesem Rezept zeige ich, wie Sie eine symmetrische Verschlüsselung von Daten erreichen. In den Rezepten 310 und 311 erfahren Sie, wie Sie Daten asymmetrisch verschlüsseln. Das Rezept 312 zeigt das Erzeugen eines Hashcodes (der nicht mehr entschlüsselbar ist).

Exkurs

Verschlüsselungs-Algorithmen setzen im Allgemeinen Schlüssel ein, deren Werte in der Verschlüsselungslogik berücksichtigt werden. Ein einfacher Algorithmus würde z. B. den Zeichencode jedes Zeichens eines zu verschlüsselnden Strings mit dem Zeichencode des korrespondierenden Schlüssel-Strings addieren und das Ergebnis speichern. Neben dem Verschlüsselungsalgorithmus bestimmt auch die Länge des Schlüssels die Qualität des Ergebnisses.

Symmetrische Verschlüsselungsverfahren setzen ein und denselben Schlüssel mit einer meist festgelegten Länge für das Ver- und das Entschlüsseln ein. Die Vorteile einer symmetrischen Verschlüsselung gegenüber einer asymmetrischen sind, dass diese schwerer geknackt werden können und das Ver- und Entschlüsseln größerer Daten erlauben. Der Nachteil einer symmetrischen Verschlüsselung ist die notwendige Weitergabe des Schlüssels: Gelangt der Schlüssel in falsche Hände, können die Daten problemlos entschlüsselt werden (sofern das Verfahren bekannt ist). Der Schlüssel muss also geheim gehalten werden (und wird deshalb als geheimer Schlüssel bezeichnet). Beispiele für symmetrische Verfahren sind RC2 (64 Bit Schlüsselgröße), DES (64 Bit), TripleDES (192 Bit), AES (256 Bit), IDEA (128 Bit) und CAST (128 oder 256 Bit).

Bei symmetrischen Verschlüsselungsverfahren werden verschiedene Chiffrier-Modi (Cipher modes) unterschieden. Block-Chiffrierungen wie CBC (Cipher Block Chaining), CFB (Cipher Feedback) und ECB (Electronic Codebook) verschlüsseln einen festen Block von Daten in einen Block gleicher Länge. Stream-Chiffrierungen verschlüsseln einzelne Bit-Gruppen typischerweise, indem diese mit dem Schlüssel bitweise mit XOR verknüpft werden.

Die Blockgröße von Block-Chiffrierungen können Sie in den .NET-Klassen einstellen. Die Arbeitsweise der von .NET unterstützten Modi werden sehr gut in der Dokumentation der `CipherMode`-Aufzählung erläutert. Bei allen Block-Chiffrierungen werden die Daten blockweise über den Schlüssel neu berechnet und im Ergebnis abgelegt. Der beliebte CBC-Modus nutzt für die Verschlüsselung zusätzlich den zuvor verschlüsselten Block. Weil für den ersten Block kein verschlüsselter Block zur Verfügung steht, arbeitet CBC mit einem so genannten Initialisierungsvektor (kurz: IV), das ist ein Block beliebiger Daten, der von außen definiert werden kann. Wenn Sie Daten im (übrigens unter .NET voreingestellten) CBC-Modus ver- und entschlüsseln, müssen Sie also neben dem Schlüssel auch den Initialisierungsvektor angeben. Die Länge des Initialisierungsvektors muss der Blockgröße entsprechen.

Der Initialisierungsvektor wird häufig dazu verwendet, verschlüsselte Daten zusätzlich zu sichern. Dazu werden einzelne Nachrichten bei gleichem Schlüssel mit unterschiedlichen Initialisierungsvektoren verschlüsselt. Der Initialisierungsvektor wird mit der Nachricht öffentlich übertragen, sodass der Empfänger nur den Schlüssel kennen muss. Ein Hacker hat es damit wesentlich schwerer, aus mehreren Nachrichten auf den verwendeten Schlüssel rückzuschließen, da das Ergebnis zweier Verschlüsselungen derselben Daten mit demselben Schlüssel, aber unterschiedlichen Initialisierungsvektoren nicht identisch ist.

Da der letzte Block meist nicht der definierten Blockgröße entspricht, wird dieser häufig mit Bytes aufgefüllt. Bei diesem so genannten Padding werden ebenfalls verschiedene Modi eingesetzt. *PKCS7* speichert dazu für jedes Byte den Wert der Anzahl der insgesamt hinzuzufügenden Bytes, der andere Modus *Zeros* speichert einfach 0-Bytes. PKCS7 ist unter .NET die Voreinstellung.

Für eine symmetrische Verschlüsselung stellt Ihnen der Namensraum `System.Security.Cryptography` einige abstrakte Basisklassen und einige davon abgeleitete konkrete Implementierungen zur Verfügung. Die Klasse `SymmetricAlgorithm` ist die gemeinsame Basisklasse aller Klassen für eine symmetrische Verschlüsselung. Eine Referenz dieses Typs reicht aus, um mit den konkreten Klassen arbeiten zu können.

Von `SymmetricAlgorithm` ist für jeden Algorithmus je eine weitere abstrakte Klasse abgeleitet und von diesen schließlich eine oder mehrere konkrete Klassen. Die Klasse `Aes` ist z. B. die abstrakte Basisklasse für die Klassen `AesCryptoServiceProvider` und `AesManaged`.

Klassen mit der Endung »Managed« sind komplett in verwaltetem Code implementiert. Klassen mit der Endung »ServiceProvider« sind lediglich Hüllen für Klassen des nativen Windows-Kryptographie-API (Crypto API). CNG-Varianten (CNG = Cryptography Next Generation, ein neues Kryptographie-Framework ab Vista und Windows Server 2003) der Klassen für symmetrische Verschlüsselungen existieren scheinbar (noch) nicht.

Über die `Create`-Methode von `SymmetricAlgorithm` können Sie eine Instanz einer Klasse für die verfügbaren Algorithmen erzeugen. Dazu übergeben Sie einen String mit dem Basisnamen (z. B. »Rijndael« für den Rijndael-Algorithmus). Die konkreten Klassen stellen eine weitere `Create`-Methode ohne Argument zur Verfügung (die natürlich eine Instanz der jeweiligen Klasse erzeugt). Da diese Klassen von `SymmetricAlgorithm` erben, besitzen sie verwirrenderweise auch die `Create`-Variante mit String-Argument. Diese ist aber sinnlos, da sie nur den für den jeweiligen Algorithmus gültigen String akzeptiert.

Tabelle 31 beschreibt die abstrakten Basisklassen und die konkreten Ableitungen für die einzelnen Algorithmen. In der letzten Spalte sind die möglichen Schlüssel- und Blockgrößen angegeben.

Basisklasse	Konkrete Klassen	Beschreibung	Schlüssel- und Blockgröße
`SymmetricAlgorithm`	Alle konkreten Verschlüsselungs-Klassen	Basisklasse für alle symmetrischen Verschlüsselungsverfahren	-
`Aes`	`AesCryptoServiceProvider` `AesManaged`	repräsentiert den AES-Algorithmus	Schlüssel: 128, 192 und 256 Bit Blockgröße: 128 Bit
`DES`	`DESCryptoServiceProvider`	repräsentiert den DES-Algorithmus	Schlüssel: 64 Bit Blockgröße: 64 Bit
`RC2`	`RC2CryptoServiceProvider`	repräsentiert den RC2-Algorithmus	Schlüssel: 40 bis 128 Bit in 8-Bit-Schritten Blockgröße: 64 Bit
`Rijndael`	`RijndaelManaged`	repräsentiert den Rijndael-Algorithmus	Schlüssel: 128, 192 und 256 Bit Blockgröße: 128, 192 und 256 Bit
`TripleDES`	`TripleDESCryptoServiceProvider`	repräsentiert den TripleDES-Algorithmus.	Schlüssel: 128 und 192 Bit Blockgröße: 64 Bit

Tabelle 31: Die .NET-Klassen für symmetrische Verschlüsselungen

Die konkreten Klassen implementieren die einzelnen Verschlüsselungs-Algorithmen. In ihrem Konstruktor erzeugen diese Klassen kryptografisch starke Werte für den Schlüssel und den Initialisierungsvektor, die dann für das Ver- und das Entschlüsseln verwendet werden können. Sie können diese aber auch mit eigenen Werten neu definieren.

Um Daten symmetrisch zu verschlüsseln, müssen Sie sich nur noch entscheiden, welchen Algorithmus Sie verwenden. Ich habe mich für dieses Rezept entschieden, Ihnen die Wahl des Algorithmus zu überlassen und eine Klasse zu entwickeln, die Daten nach allen genannten Algorithmen ver- und entschlüsseln kann. Dieses Vorhaben wurde enorm dadurch erleichtert, dass alle Klassen für symmetrische Verschlüsselungen von der Basisklasse `SymmetricAlgorithm` abgeleitet sind und keine Klasse zu den geerbten Elementen wesentliche, spezifische Elemente hinzufügt. Die Arbeit an dieser Klasse war übrigens nicht allzu einfach und hat mittlerweile mehrere Tage gedauert, weil ich (wie in den meisten anderen Rezepten dieses Buchs auch) eigene Ideen verwirklichen wollte. Dafür ist die Klasse aber auch sehr flexibel einsetzbar ☺.

Um diese Klasse kompilieren zu können, müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Security.Cryptography` importieren.

Zur Auswahl des Algorithmus habe ich zunächst eine Aufzählung `SymmetricEncryptAlgorithm` deklariert, die Konstanten für die unterstützten Algorithmen zur Verfügung stellt:

```
public enum SymmetricEncryptAlgorithm
{
   AES,
   DES,
   TrippleDES,
   RC2,
   Rijndael
}
```

Listing 577: Aufzählung für den Verschlüsselungsalgorithmus

Instanzen der Klasse `SymmetricEncryptor` übernehmen die Ver- und Entschlüsselung. Die Klasse enthält zunächst ein privates Feld vom Typ `SymmetricAlgorithm`, das die Instanz des Verschlüsselungsobjekts verwaltet, und ein Feld, das den Namen des aktuell gewählten Algorithmus speichert (und das in Fehlermeldungen verwendet wird). Dann wird in dem privaten Feld `stringEncoding` die Codierung für die Umwandlung von Byte-Arrays in Strings und zurück gespeichert. Eine solche Umwandlung wird z. B. für die Verwaltung des Schlüssels und des Initialisierungsvektors als String benötigt. Bei der verwendeten Codierung muss es sich zwingend um eine 8-Bit-Codierung handeln, da andere Codierungen (wie UTF-8, mit einer variablen Byteanzahl pro Zeichen, oder UFT-32, mit zwei Byte pro Zeichen) zu Problemen beim Entschlüsseln der Daten führen. Ich setze hier die im Internet verwendete 8-Bit-Codierung ISO-8859-1 ein, die mit UTF-8 in dem ersten Byte eines Zeichens kompatibel ist. Sie müssen nun aber nicht befürchten, dass die Methoden zum Ver- und Entschlüsseln nicht mit Unicode-Strings oder -Dateien zurechtkommen. Unicode-Daten werden einwandfrei ver- und entschlüsselt. Ich erläutere bei den entsprechenden Eigenschaften und Methoden noch, wo und warum ich ISO-8859-1 eingesetzt habe.

Dem Konstruktor wird am ersten Argument der zu verwendende Algorithmus übergeben. Der Konstruktor erzeugt eine dazu passende Instanz einer der verfügbaren Klassen für symmetrische Verschlüsselungsverfahren und speichert die Referenz in dem Feld `SymmetricEncryptor`. Außerdem wird der Name des Verschlüsselungsverfahrens in dem dafür vorgesehenen Feld abgelegt:

```
public class SymmetricEncryptor
{
   /* Verwaltet die Instanz des Verschlüsselers */
   private SymmetricAlgorithm encryptor;

   /* Verwaltet den Namen des Verschlüsselungs-Algorithmus */
   private string algorithmName;

   /* Die Codierung für das Konvertieren von Strings in Byte-Arrays
      und umgekehrt. Es muss sich dabei zwingend um eine 8-Bit-Codierung
      handeln, da es ansonsten Probleme mit dem Entschlüsseln von
      verschlüsselten Strings gibt. */
   private Encoding stringEncoding = Encoding.GetEncoding("ISO-8859-1");

   /* Konstruktor  */
   public SymmetricEncryptor(SymmetricEncryptAlgorithm algorithm)
   {
      switch (algorithm)
      {
         case SymmetricEncryptAlgorithm.AES:
            this.encryptor = new AesManaged();
            this.algorithmName = "AES";
            break;

         case SymmetricEncryptAlgorithm.DES:
            this.encryptor = new DESCryptoServiceProvider();
            this.algorithmName = "DES";
            break;

         case SymmetricEncryptAlgorithm.TrippleDES:
            this.encryptor = new TripleDESCryptoServiceProvider();
            this.algorithmName = "TripleDES";
            break;

         case SymmetricEncryptAlgorithm.RC2:
            this.encryptor = new RC2CryptoServiceProvider();
            this.algorithmName = "RC2";
            break;
```

Listing 578: Die privaten Felder und der Konstruktor der Klasse SymmetricEncryptor

```
            case SymmetricEncryptAlgorithm.Rijndael:
               this.encryptor = new RijndaelManaged();
               this.algorithmName = "Rijndael";
               break;
         }
      }
```

Listing 578: Die privaten Felder und der Konstruktor der Klasse SymmetricEncryptor (Forts.)

Nun folgen einige Eigenschaften, über die Sie den zu verwendenden Chiffrier-Modus und das Padding definieren können.

```
/* Gibt den Chiffrier-Modus an */
public CipherMode CipherMode
{
   get { return this.encryptor.Mode; }
   set { this.encryptor.Mode = value; }
}

/* Gibt den Padding-Modus an */
public PaddingMode PaddingMode
{
   get { return this.encryptor.Padding; }
   set { this.encryptor.Padding = value; }
}
```

Listing 579: Eigenschaften zum Einstellen des Chiffrier-Modus und des Padding

Die Eigenschaft `BlockSize` gibt die Blockgröße an. Da die gleichnamige Eigenschaft des Verschlüsselungs-Objekts eine nicht sehr sprechende Ausnahme wirft, wenn eine ungültige Größe angegeben wird, überprüft meine Eigenschaft vor dem Schreiben, ob die geschriebene Blockgröße in Ordnung ist. Die Blockgrößen werden über die eigene Eigenschaft `LegalBlockSizes` ermittelt, die die gleichnamige Eigenschaft des Verschlüsselungs-Objekts verwendet.

Die dazu auszulesende Auflistung `LegalBlockSizes` liefert ein Array von `KeySize`-Objekten. Diese geben über die Eigenschaft `MinSize` den kleinsten und über `MaxSize` den größten möglichen Schlüssel an. Die Eigenschaft `SkipSize` definiert den Abstand zwischen zwei Schlüsselgrößen. Typische Werte sind z. B. `MinSize` = 128, `MaxSize` = 256 und `SkipSize` = 128. Es kann aber auch vorkommen, dass `MinSize` gleich `MaxSize` ist.

Um die Auswertung der möglichen Blockgrößen nach außen einfacher zu machen, liefert die Eigenschaft `LegalBlockSizes` ein `int`-Array und die Eigenschaft `LegalBlockSizesString` zudem die gültigen Blockgrößen als String.

```
/* Liefert die möglichen Blockgrößen in Bit */
public int[] LegalBlockSizes
{
   get
   {
      List<int> validBlockSizes = new List<int>();
      foreach (var blockSize in this.encryptor.LegalBlockSizes)
      {
         if (blockSize.MinSize == blockSize.MaxSize)
         {
            validBlockSizes.Add(blockSize.MinSize);
         }
         else
         {
            for (int i = blockSize.MinSize; i <= blockSize.MaxSize;
               i += blockSize.SkipSize)
            {
               validBlockSizes.Add(i);
            }
         }
      }
      validBlockSizes.Sort();
      return validBlockSizes.ToArray();
   }
}

/* Liefert die möglichen Blockgrößen in Bit als String */
public string LegalBlockSizesString
{
   get
   {
      int[] legalBlockSizes = this.LegalBlockSizes;
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < legalBlockSizes.Length; i++)
      {
         if (i > 0)
         {
            result.Append(", " + legalBlockSizes[i]);
         }
         else
         {
            result.Append(legalBlockSizes[i].ToString());
         }
      }
```

Listing 580: Eigenschaften zur Definition der Blockgröße

```
            return result.ToString();
        }
    }

    /* Verwaltet die zu verwendende Blockgröße */
    public int BlockSize
    {
        get { return this.encryptor.BlockSize; }
        set
        {
            // Die geschriebene Blockgröße überprüfen
            int[] legalBlockSizes = this.LegalBlockSizes;
            bool blockSizeOK = false;
            foreach (var legalBlockSize in legalBlockSizes)
            {
                if (value == legalBlockSize)
                {
                    blockSizeOK = true;
                    break;
                }
            }
            if (blockSizeOK)
            {
                this.encryptor.BlockSize = value;
            }
            else
            {
                // Ungültige Blockgröße: Ausnahme mit erweiterten Informationen
                // werfen
                throw new CryptographicException("Die übergebene Blockgröße " +
                    "ist mit " + (value * 8) + " Bit für den " +
                    this.algorithmName + "-Algorithmus ungültig. Erlaubt " +
                    "sind die folgenden Größen: " +
                    this.LegalBlockSizesString + ".");
            }
        }
    }
```

Listing 580: Eigenschaften zur Definition der Blockgröße (Forts.)

Der Konstruktor der verwendeten Verschlüsselungsklasse erzeugt automatisch einen Zufalls-Schlüssel. Damit Sie diesen lesen, aber auch nach der Erzeugung der Instanz mit einem eigenen Schlüssel überschreiben können, habe ich die Eigenschaft `Key` implementiert, die auf die gleichnamige Eigenschaft im Verschlüsselungs-Objekt zugreift. Zusätzlich dazu habe ich die Eigenschaft `KeyString` definiert, die den Schlüssel als String verwaltet. Dabei musste ich das Problem lösen, dass der originale Byte-Array-Schlüssel in einen String konvertiert werden muss.

Das Problem dabei ist (wie sehr häufig bei Strings) die Codierung. Strings werden in .NET-Programmen in der UTF-8-Codierung gespeichert. Würde `KeyString` den übergebenen String über `System.Text.Encoding.UTF8.GetBytes` bzw. `GetString` decodieren bzw. codieren, würde das hohe Byte jedes Unicode-Zeichens mit in das Byte-Array übertragen werden. Da die meisten Strings nur Zeichen im Bereich bis 0x00FF (255) enthalten, wäre jedes zweite Byte in diesem Falle 0. Der Schlüssel wäre also eigentlich nur die Hälfte wert. Deshalb habe ich mich entschlossen, als Codierung die zum 8-Bit-Teil des Unicode-Zeichensatzes kompatible Codierung ISO-8859-1 und damit nur das niedrige Byte jedes Zeichens als Schlüssel zu verwenden. So können Sie alle 8-Bit-Zeichen des Unicode-Zeichensatzes für den Schlüssel verwenden und besitzen damit auch alle Möglichkeiten, die der Schlüssel zur Verfügung stellt (eben 8 Bit pro Byte ☺). Sie können lediglich keine Strings mit Unicode-Zeichen größer 255 als Schlüssel verwenden.

Um sicherzustellen, dass nicht aus Versehen Schlüssel mit Zeichen übergeben werden, die größer sind als 0x00FF, prüft `KeyString` beim Schreiben jedes Zeichen daraufhin ab und generiert im Fehlerfall eine Ausnahme vom Typ `CryptographicException`.

Da die verschiedenen Verschlüsselungsverfahren mit unterschiedlichen Schlüsselgrößen arbeiten, überprüft `Key` (die auch von `KeyString` verwendet wird) beim Setzen des Schlüssels zudem über die `ValidKeySize`-Methode des Verschlüsselungs-Objekts, ob die Schlüssellänge in Ordnung ist, und generiert im Fehlerfall eine Ausnahme. Ich habe mich hierbei nicht auf die Ausnahme verlassen, die die `Key`-Eigenschaft des Verschlüsselungs-Objekts wirft, wenn der Schlüssel ungültig ist, weil diese nur die allgemeine Fehlermeldung ausgibt, dass der Schlüssel ungültig ist. Meine Ausnahme liefert zudem die Information, welche Schlüsselgrößen erlaubt sind. Dazu habe ich die Eigenschaften `LegalKeySizes` und `LegalKeySizesString` entwickelt, die (wie schon bei der Blockgröße) die möglichen Schlüsselgrößen in einer besser auswertbaren Form zurückgeben.

```
/* Liefert die möglichen Schlüsselgrößen in Bit */
public int[] LegalKeySizes
{
   get
   {
      List<int> validKeySizes = new List<int>();
      foreach (var keySize in this.encryptor.LegalKeySizes)
      {
         if (keySize.MinSize == keySize.MaxSize)
         {
            validKeySizes.Add(keySize.MinSize);
         }
         else
         {
            for (int i = keySize.MinSize; i <= keySize.MaxSize;
               i += keySize.SkipSize)
```

Listing 581: Eigenschaften zum Lesen und Setzen des Schlüssels und zur Ermittlung der möglichen Schlüsselgrößen

```
            {
               validKeySizes.Add(i);
            }
         }
      }
      validKeySizes.Sort();
      return validKeySizes.ToArray();
   }
}

/* Liefert die möglichen Schlüsselgrößen in Bit als String */
public string LegalKeySizesString
{
   get
   {
      int[] legalKeySizes = this.LegalKeySizes;
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < legalKeySizes.Length; i++)
      {
         if (i > 0)
         {
            result.Append(", " + legalKeySizes[i]);
         }
         else
         {
            result.Append(legalKeySizes[i].ToString());
         }
      }
      return result.ToString();
   }
}

/* Verwaltet den Schlüssel */
public byte[] Key
{
   get
   {
      return this.encryptor.Key;
   }
   set
   {
      // Den übergebenen Schlüssel überprüfen
      if (this.encryptor.ValidKeySize(value.Length * 8) == false)
      {
```

Listing 581: Eigenschaften zum Lesen und Setzen des Schlüssels und zur Ermittlung der möglichen Schlüsselgrößen (Forts.)

```
            // Ungültiger Schlüssel: Ausnahme mit erweiterten Informationen
            // werfen
            throw new CryptographicException("Der übergebene Schlüssel " +
               "ist mit " + (value.Length * 8) + " Bit für den " +
               this.algorithmName + "-Algorithmus ungültig. Erlaubt " +
               "sind die folgenden Größen: " +
               this.LegalKeySizesString + ".");
         }

         this.encryptor.Key = value;
      }
   }

   /* Verwaltet den Schlüssel als String */
   public string KeyString
   {
      get
      {
         return this.stringEncoding.GetString(this.Key);
      }

      set
      {
         // Auf Unicode-Zeichen größer 0x00FF überprüfen
         for (int i = 0; i < value.Length; i++)
         {
            if ((int)value[i] > 255)
            {
               throw new CryptographicException("Der übergebene " +
                  "Schlüssel enthält mindestens ein Unicode-Zeichen, " +
                  "das größer ist als 0x00FF (255): " + value[i] +
                  " (" + (int)value[i] + "). Unterstützt werden lediglich " +
                  "8-Bit-Unicode-Zeichen");
            }
         }

         // Den Schlüssel setzen
         this.Key = this.stringEncoding.GetBytes(value);
      }
   }
```

Listing 581: Eigenschaften zum Lesen und Setzen des Schlüssels und zur Ermittlung der möglichen Schlüsselgrößen (Forts.)

Wenn Sie Daten im CBC-Modus verschlüsseln (was die Voreinstellung ist!), benötigt das Verschlüsselungsobjekt einen Initialisierungsvektor aus Bytes, dessen Größe der aktuell eingestellten Blockgröße entspricht. Der Konstruktor der verwendeten Verschlüsselungsklassen erzeugt diesen automatisch. Ähnlich wie bei der `Key`-Eigenschaft müssen Sie diesen lesen und

auch schreiben können. Die Eigenschaft `IV` erlaubt deswegen das Lesen und Schreiben des Initialisierungsvektors. Wie bei `Key` habe ich zudem die Eigenschaft `IVString` definiert, die einen String als Initialisierungsvektor erlaubt. Diese Eigenschaft stellt sicher, dass nur der 8-Bit-Bereich der Unicode-Zeichen verwendet wird. Der übergebene String wird deshalb wieder daraufhin überprüft, ob er Zeichen speichert, die größer sind als 0x00FF.

Da die Größe des Initialisierungsvektors der aktuellen Blockgröße entsprechen muss und im Fehlerfall eine Ausnahme erst beim Ver- oder Entschlüsseln generiert werden würde, überprüft `IV` die Größe des Initialisierungsvektors gegen die Eigenschaft `BlockSize` und wirft im negativen Fall eine aussagekräftige Ausnahme.

```
public byte[] IV
{
   get
   {
      return this.encryptor.IV;
   }
   set
   {
      // Den übergebenen Initialisierungsvektor überprüfen
      if ((value.Length * 8) != this.encryptor.BlockSize)
      {
         // Ungültiger Initialisierungsvektor: Ausnahme mit erweiterten
         // Informationen werfen
         throw new CryptographicException("Der übergebene " +
            "Initialisierungsvektor ist mit " + (value.Length * 8) +
            " Bit für den " + this.algorithmName + "-Algorithmus " +
            "ungültig. Die Länge des Initialisierungsvektors muss " +
            "durch die Blockgröße (aktuell " + this.encryptor.BlockSize +
            ") ohne Rest teilbar sein");
      }

      // Den Initialisierungsvektor setzen
      this.encryptor.IV = value;
   }
}
```

Listing 582: Eigenschaft zum Zugriff auf den Initialisierungsvektor

Die Eigenschaft `IVString` erlaubt die Definition des Initialisierungsvektors als String:

```
public string IVString
{
   get
   {
      return this.stringEncoding.GetString(this.IV);
```

Listing 583: Eigenschaft zur Definition des Initialisierungsvektors als String

```
    }

    set
    {
       // Auf Unicode-Zeichen größer 0x00FF überprüfen
       for (int i = 0; i < value.Length; i++)
       {
          if ((int)value[i] > 255)
          {
             throw new CryptographicException("Der übergebene " +
                "Initialisierungsvektor enthält mindestens ein " +
                "Unicode-Zeichen, das größer ist als 0x00FF (255): " +
                value[i] + " (" + (int)value[i] + "). Unterstützt " +
                "werden lediglich 8-Bit-Unicode-Zeichen");
          }
       }

       // Den Initialisierungsvektor setzen
       this.IV = this.stringEncoding.GetBytes(value);
    }
}
```

Listing 583: Eigenschaft zur Definition des Initialisierungsvektors als String (Forts.)

Als kleine Zugabe habe ich die Methoden zur Erzeugung eines neuen zufälligen Schlüssels und Initialisierungsvektors übernommen:

```
/* Erzeugt einen zufälligen Schlüssel */
public byte[] GenerateKey()
{
   this.encryptor.GenerateKey();
   return this.Key;
}

/* Erzeugt einen zufälligen Schlüssel */
public string GenerateKeyString()
{
   this.encryptor.GenerateKey();
   return this.KeyString;
}

/* Erzeugt einen zufälligen Initialisierungsvektor */
public byte[] GenerateIV()
{
   this.encryptor.GenerateIV();
```

Listing 584: Methoden zur Erzeugung eines zufälligen Schlüssels und Initialisierungsvektors

```
    return this.IV;
}

/* Erzeugt einen zufälligen Initialisierungsvektor */
public string GenerateIVString()
{
    this.encryptor.GenerateIV();
    return this.IVString;
}
```

Listing 584: Methoden zur Erzeugung eines zufälligen Schlüssels und Initialisierungsvektors (Forts.)

Das Ver- und Entschlüsseln

Nun folgt der interessante Teil: das Ver- und Entschlüsseln. Die erste Variante der Methode `Encrypt` verschlüsselt einen Stream in einen anderen Stream. `Encrypt` arbeitet mit Streams, weil die Verschlüsselungs-Objekte auch damit arbeiten und weil diese Methode dadurch enorm flexibel wird. So können Sie z. B. `FileStream`-Objekte verwenden, um Dateien zu verschlüsseln, oder Daten aus einem `MemoryStream`-Objekt in einen `NetworkStream` schreiben um diese über das Internet oder ein normales Netz zu versenden.

Zum Verschlüsseln erzeugt `Encrypt` eine Instanz der Klasse `CryptoStream`. Am ersten Argument wird der Stream übergeben, in den das `CryptoStream`-Objekt die verschlüsselten Daten schreibt. Am zweiten Argument übergibt `Encrypt` ein Objekt, das die Schnittstelle `ICryptoTransform` implementiert, über die die Verschlüsselung vorgenommen wird. Dieses Objekt wird über die Methode `CreateEncryptor` des aktuellen Verschlüsselungs-Objekts ermittelt. Am letzten Argument wird dann noch festgelegt, dass der `CryptoStream` in den Zielstream schreiben soll.

Danach werden die Daten des übergebenen Quell-Streams in den Crypto-Stream geschrieben, wobei das Schreiben blockweise erfolgt, damit der Arbeitsspeicher bei großen Streams nicht überlastet wird. Das `CryptoStream`-Objekt sorgt automatisch dafür, dass die geschriebenen Daten über das `ICryptoTransform`-Objekt verschlüsselt und in den Ziel-Stream geschrieben werden. Nach dem Schreiben wird über die `FlushFinalBlock`-Methode des Crypto-Streams nur noch dafür gesorgt, dass die noch im Stream vorhandenen Restdaten verarbeitet werden.

Der Crypto-Stream wird übrigens explizit nicht geschlossen. Der Grund dafür ist, dass ein Aufruf der `Close`-Methode bewirkt, dass (nach eigenen empirischen Ermittlungen) auch der Ziel-Stream (der dem Crypto-Stream im Konstruktor übergeben wurde) geschlossen wird. `Encrypt` kann aber nicht voraussetzen, dass der Zielstream nicht mehr benötigt wird. Wird z. B. ein `MemoryStream` übergeben, ist es nach dem Verschlüsseln notwendig, den Stream auf die erste Position zu setzen und auszulesen.

```
public void Encrypt(Stream sourceStream, Stream destStream)
{
    // CryptoStream zum Verschlüsseln erzeugen. Als Transformations-
    // Objekt wird das Verschlüssel-Objekt der aktuellen
```

Listing 585: Methode zum Verschlüsseln eines Streams

```
    // SymmetricAlgorithm-Instanz übergeben
    CryptoStream cryptoStream = new CryptoStream(destStream,
       this.encryptor.CreateEncryptor(), CryptoStreamMode.Write);

    // Die Rohdaten blockweise in den CryptoStream schreiben (der diese
    // verschlüsselt in den Ziel-Stream schreibt)
    int bytesRead = 0;
    byte[] buffer = new byte[1024];
    do
    {
       bytesRead = sourceStream.Read(buffer, 0, 1024);
       cryptoStream.Write(buffer, 0, bytesRead);
    } while (bytesRead > 0);

    // Den Zielstream aktualisieren und den Puffer löschen
    cryptoStream.FlushFinalBlock();

    // Der CryptoStream darf hier nicht geschlossen werden, da
    // dieser den Ziel-Stream ansonsten auch schließt
}
```

Listing 585: Methode zum Verschlüsseln eines Streams (Forts.)

Die erste Variante der Methode `Decrypt` entschlüsselt einen Stream in einen anderen Stream. `Decrypt` erzeugt dazu wieder eine `CryptoStream`-Instanz, der aber am ersten Argument nun der Quell-Stream übergeben wird. Am zweiten Argument wird das `ICryptoTransform`-Objekt übergeben, das die `CreateDecryptor`-Methode des aktuellen Verschlüsselungs-Objekts zurückgibt. Das dritte Argument bestimmt mit `CryptoStreamMode.Read`, dass die Daten des am ersten Argument übergebenen Streams gelesen werden sollen.

Dann werden die Daten blockweise aus dem Crypto-Stream in einen Byte-Puffer gelesen (wieder um den Arbeitsspeicher nicht zu sehr zu belasten) und in den Ergebnis-Stream geschrieben. Beim Lesen der Daten werden diese automatisch über das `ICryptoTransform`-Objekt entschlüsselt.

Der Crypto-Stream darf wieder nicht geschlossen werden, da ansonsten auch der Quell-Stream geschlossen wird.


```
public void Decrypt(Stream sourceStream, Stream destStream)
{
    // CryptoStream zum Entschlüsseln erzeugen. Als Transformations-
    // Objekt wird das Entschlüssel-Objekt der aktuellen
    // SymmetricAlgorithm-Instanz übergeben
    CryptoStream cryptoStream = new CryptoStream(sourceStream,
       this.encryptor.CreateDecryptor(), CryptoStreamMode.Read);
```

Listing 586: Methode zum Entschlüsseln eines Streams

```
    // Daten blockweise einlesen
    int bytesRead = 0;
    byte[] buffer = new Byte[1024];
    do
    {
       bytesRead = cryptoStream.Read(buffer, 0, 1024);
       destStream.Write(buffer, 0, bytesRead);
    } while (bytesRead > 0);

    // Der CryptoStream darf hier nicht geschlossen werden, da
    // dieser den Quell-Stream ansonsten auch schließt
}
```

Listing 586: Methode zum Entschlüsseln eines Streams (Forts.)

Eigentlich wäre die Klasse nun komplett. Ich habe aber noch zwei Methoden zum Ver- und Entschlüsseln von Strings hinzugefügt, da die dazu notwendige Umwandlung in einen `MemoryStream` recht aufwändig ist. Beide Methoden erzeugen `MemoryStream`-Objekte und geben diese einfach an die Stream-Variante der entsprechenden Methode weiter. Dabei habe ich in `Encrypt` die eingangs erwähnte 8-Bit-Codierung verwendet um das verschlüsselte Byte-Array als String zurückzugeben und in `Decrypt` um den übergebenen verschlüsselten (8-Bit-) String in ein Byte-Array umzuwandeln. Der `Encrypt` übergebene String wird allerdings über die UTF8-Codierung in einen `MemoryStream` überführt, in `Decrypt` nutze ich diese Codierung zur Rückgabe des entschlüsselten Strings. Damit können Sie auch Strings mit Unicode-Zeichen größer 255 ver- und entschlüsseln (wie das Beispiel zu diesem Rezept beweist).

```
/* Verschlüsselt einen String */
public string Encrypt(string source)
{
   // MemoryStreams für die Daten erzeugen
   MemoryStream sourceStream =
      new MemoryStream(Encoding.UTF8.GetBytes(source));
   MemoryStream destStream = new MemoryStream();

   // Daten verschlüsseln
   sourceStream.Position = 0;
   this.Encrypt(sourceStream, destStream);

   // Ergebnis auslesen
   destStream.Position = 0;
   byte[] encryptedBytes = destStream.ToArray();
```

Listing 587: Methoden zum Ver- und Entschlüsseln von Strings und Abschluss der SymmetricEncryptor-Klasse

```
      // Streams schließen
      sourceStream.Close();
      destStream.Close();

      // String zurückgeben
      return this.stringEncoding.GetString(encryptedBytes);
   }

   /* Entschlüsselt einen String */
   public string Decrypt(string source)
   {
      // MemoryStreams für die Daten erzeugen
      MemoryStream sourceStream =
         new MemoryStream(this.stringEncoding.GetBytes(source));
      MemoryStream destStream = new MemoryStream();

      // Daten entschlüsseln
      this.Decrypt(sourceStream, destStream);

      // Ergebnis auslesen
      destStream.Position = 0;
      byte[] encryptedBytes = destStream.ToArray();

      // Streams schließen
      sourceStream.Close();
      destStream.Close();

      // String zurückgeben
      return Encoding.UTF8.GetString(encryptedBytes);
   }
}
```

Listing 587: Methoden zum Ver- und Entschlüsseln von Strings und Abschluss der SymmetricEncryptor-Klasse (Forts.)

Die Anwendung der Klasse SymmetricEncryptor

Zum Ver- und Entschlüsseln erzeugen Sie eine Instanz der Klasse `SymmetricEncryptor`. Wenn Sie den Schlüssel und den Initialisierungsvektor nicht selbst definieren wollen, können Sie beim Verschlüsseln die zufällig erzeugten Werte aus den Eigenschaften `Key`, `KeyString` und `IV` bzw. `IVString` auslesen. Diese Daten sollten Sie auf jeden Fall irgendwo speichern, da Sie diese Werte beim Entschlüsseln wieder benötigen.

Listing 588 zeigt die Vorgehensweise am Beispiel. Das Beispiel ver- und entschlüsselt eine Datei nach dem Rijndael-Algorithmus und setzt dazu eigene Schlüssel- und Initialisierungsvektor-Werte ein. Auf eine Ausnahmebehandlung habe ich im Beispiel aus Übersichtsgründen verzichtet.

```
// SymmetricEncryptor-Instanz erzeugen und den eigenen Schlüssel und
// Initialisierungsvektor übergeben
SymmetricEncryptor encryptor =
   new SymmetricEncryptor(SymmetricEncryptAlgorithm.Rijndael);
encryptor.KeyString = "dfdOß3q3ßdsäfßq3#sfjbxya<xsd-:,?";
encryptor.IVString = "46O?-B,-7,kerkh-";

// Eine Datei verschlüsseln
FileStream sourceStream = new FileStream("c:\Hitchhiker.txt", FileMode.Open,
   FileAccess.Read);
FileStream destStream = new FileStream("c:\Hitchhiker_Encrypted.txt",
   FileMode.Create, FileAccess.Write);
encryptor.Encrypt(sourceStream, destStream);
sourceStream.Close();
destStream.Close();

// Die Datei wieder entschlüsseln
sourceStream = new FileStream("c:\Hitchhiker_Encrypted.txt", FileMode.Open,
   FileAccess.Read);
destStream = new FileStream("c:\Hitchhiker.txt", FileMode.Create,
   FileAccess.Write);
encryptor.Decrypt(sourceStream, destStream);
sourceStream.Close();
destStream.Close();
```

Listing 588: Beispiel für die Anwendung der SymmetricEncryptor-Klasse

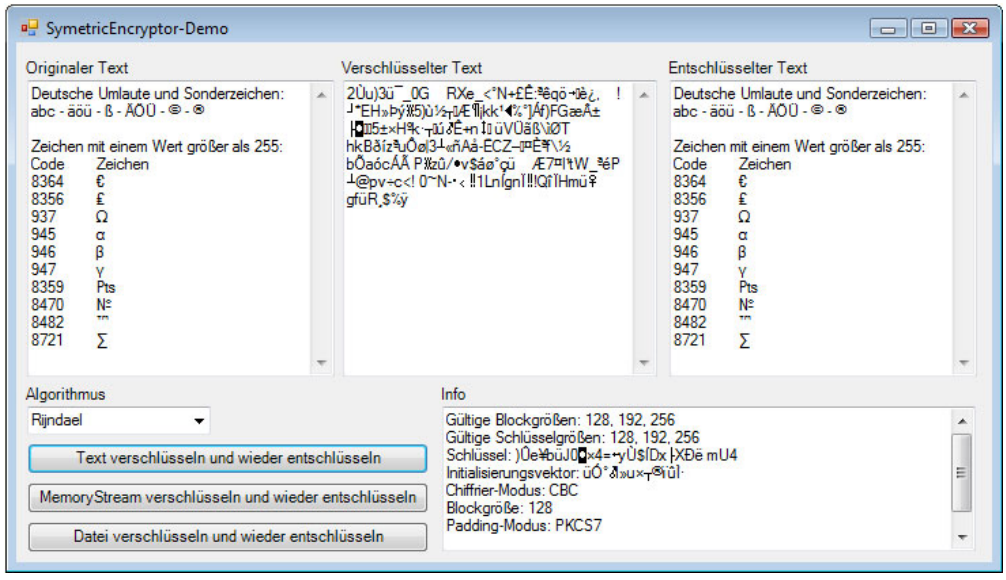


Abbildung 170: Das Beispielprogramm zu diesem Rezept

Bei der Verwendung der `SymmetricEncryptor`-Klasse müssen Sie nur wenig berücksichtigen:

- Wenn Sie Streams ver- oder entschlüsseln, stehen die Streams nach dem Ver- bzw. Entschlüsseln am Ende. Wenn Sie die Streams weiter verarbeiten wollen, müssen Sie diese über *Stream*.`Seek`(0, `SeekOrigin`.`Begin`) zunächst wieder auf den Anfang setzen. Außerdem sollten Sie die Streams nach der Verarbeitung schließen.
- Zum Entschlüsseln benötigen Sie den gleichen Schlüssel und Initialisierungsvektor, der beim Verschlüsseln verwendet wurde. Sie sollten für beide zufällig ermittelte Werte verwenden, um eine höchstmögliche Sicherheit zu erreichen. Die beim Verschlüsseln verwendeten Werte müssen Sie natürlich für das Entschlüsseln verfügbar haben.

Ein Performancetest der `SymmetricEncryptor`-Klasse hat übrigens ergeben, dass diese recht schnell arbeitet und kein Speicherloch aufweist (was eventuell durch das nicht erfolgte Schließen des Crypto-Streams verursacht werden könnte). Die Verschlüsselung einer Textdatei mit 3.656 Byte Größe benötigte auf meinem (alten) AMD-64-4400-Rechner im Durchschnitt ca. 0,00071 Sekunden, das Entschlüsseln durchschnittlich 0,00065 Sekunden. Bei 10.000 Durchläufen wurde der Speicher anfangs etwas vergrößert, blieb dann aber stabil und wurde sogar auch wieder verkleinert. Den Test finden Sie neben dem Beispiel zu diesem Rezept in den Buchbeispielen.

310 Asymmetrisches Verschlüsseln mit RSA

Beim asymmetrischen Verschlüsseln von Daten werden zwei unterschiedliche Schlüssel meist mehr oder weniger variabler Länge für das Ver- und das Entschlüsseln der Daten verwendet. Die Schlüssel stehen in einer mathematischen Beziehung, die meist auf Primzahlen basiert, und bilden ein Schlüsselpaar. Einer der Schlüssel wird geheim gehalten und daher als privater Schlüssel bezeichnet. Der andere Schlüssel wird als öffentlicher Schlüssel bezeichnet und kann problemlos öffentlich, z. B. über E-Mail, versendet werden. Diese Verfahren werden deshalb auch als Public-Key-Verfahren bezeichnet.

Asymmetrische Verschlüsselungsverfahren erlauben, Daten über den öffentlichen Schlüssel zu verschlüsseln, die dann nur über den privaten Schlüssel wieder entschlüsselt werden können. Umgekehrt können Daten, die über den privaten Schlüssel verschlüsselt wurden, nur über den öffentlichen Schlüssel wieder entschlüsselt werden (deswegen heißt das Ganze auch »asymmetrisch« ☺).

Beim sicheren Versenden von Daten muss der Sender also nur den öffentlichen Schlüssel des Empfängers kennen, um die Daten so verschlüsseln zu können, dass nur der Empfänger (und – je nach Algorithmus und Schlüssellänge – ein gut ausgestatteter Hacker) die Daten wieder entschlüsseln kann.

Die andere Richtung, also das Verschlüsseln mit dem privaten und das Entschlüsseln mit dem öffentlichen Schlüssel, wird für digitale Signaturen verwendet. Digitale Signaturen sollen sicherstellen, dass unverschlüsselt gesendete Daten beim Empfänger nicht ausgewertet werden, wenn diese von einem »Man in the Middle«[1] ausgelesen und in veränderter Form weitergesendet wurden.

1. einem Angreifer, der sich in die Kommunikation zwischen zwei Partnern eingeschleust hat.

Dazu wird aus den zu übertragenden Daten ein Hashcode berechnet. Dieser wird mit dem privaten Schlüssel des Senders verschlüsselt. Der sich daraus ergebende Wert wird als Signatur mit den Daten übertragen.

Der Empfänger entschlüsselt die Signatur mit dem öffentlichen Schlüssel des Senders und vergleicht den so ermittelten Hashcode mit einem selbst berechneten. Sind beide gleich, kann davon ausgegangen werden, dass die Daten nicht verändert wurden.

Das asymmetrische Verschlüsseln ist in .NET erstaunlich einfach. Dazu stehen zunächst einige abstrakte Klassen zur Verfügung. Die von der gemeinsamen Basisklasse `AsymmetricAlgorithm` abgeleiteten abstrakten Klassen besitzen wie schon bei der symmetrischen Verschlüsselung eine `Create`-Methode, die die eine Instanz der »Standardimplementierung« zurückgibt.

Basisklasse	Konkrete Klasse(n)	Beschreibung	Schlüssellänge
`AsymmetricAlgorithm`	Alle konkreten Klassen für die asymmetrische Verschlüsselung	Basisklasse für alle Klassen, die eine asymmetrische Verschlüsselung implementieren	-
`DSA`	`DSACryptoServiceProvider`	repräsentiert den DSA-Algorithmus (Digital Signature Algorithm). DSA wird ausschließlich für digitale Signaturen verwendet.	512 bis 1024 Bit in 64-Bit-Schritten
`RSA`	`RSACryptoServiceProvider`	repräsentiert den RSA-Algorithmus. RSA ist sicherer als DSA, wurde aber (mit kleinen Schlüsselgrößen) bereits mehrfach geknackt (Siehe *citeseer.ist.psu.edu/514527.html*). Für eine hohe Sicherheit sind große Schlüssel ab 1.024 Bit zu empfehlen. 2.048-Bit-Schlüssel sind für die Zukunft wohl ausreichend groß dimensioniert.	384 bis und 16384 Bit in 8-Bit-Schritten wenn der Microsoft Enhanced Cryptographic Provider installiert. Ansonsten nur bis 512 Bit.
`ECDiffieHellman`	`ECDiffieHellmanCng`	repräsentiert den Elliptic Curve Diffie-Hellman-Algorithmus (ECDH). Dieser schon recht alte Algorithmus ist schwierig zu knacken, weil dazu eine logarithmische Rückrechnung vorgenommen werden muss.	256, 384 und 521 Bit
`ECDsa`	`ECDsaCng`	repräsentiert den ECDSA-Algorithmus (Elliptic Curve Digital Signature Algorithm).	256 und 384 Bit

Tabelle 32: Die Klassen für asymmetrische Verschlüsselungen

Achtung

Beachten Sie, dass die »Cng«-Varianten zum »Cryptography Next Generation Framework« gehören, das erst ab Vista und Windows Server 2003 zur Verfügung steht (die Dokumentation behauptet allerdings, die entsprechenden Klassen würden auch unter Windows XP SP2 zur Verfügung stehen, aber auf meinen Windows-XP-SP2 erhielt ich beim Versuch der Instanzierung eine `TargetInvocationException`).

Hinweis

Die Klassen zur asymmetrischen Verschlüsselung sind leider recht »durcheinander«. Die abstrakten Basisklassen stellen entweder keine Methode zur Ver- und Entschlüsselung zur Verfügung (weswegen die Verwendung der `Create`-Methode auf einer abstrakten Basisklasse keinen Sinn macht), oder diese, wie im Fall von `RSA`, eine `NotSupportedException` werfen. Damit sind Sie in der Praxis schon einmal gezwungen, mit den konkreten Klassen zu arbeiten.

Zum anderen verwenden die `ECDiffieHellman`- und die `ECDsa`-Klasse eine andere Grundtechnik als `DSA` und `RSA`. Auf `ECDiffieHellman` gehe ich im Rezept 311 ein.

Mit RSA verschlüsseln

Da DSA nur für das digitale Signieren vorgesehen ist und ECDH eine andere Technik verwendet, behandelt dieses Rezept das Ver- und Entschlüsseln mit dem relativ sicheren RSA (wenn Sie Schlüsselgrößen ab 124 Bit einsetzen).

Dazu müssen Sie zunächst einmal die Schlüssel erzeugen, die Sie verwenden wollen. Wie schon bei der symmetrischen Verschlüsselung generiert die jeweilige Klasse beim Erzeugen einer Instanz Zufalls-Schlüssel (allerdings just in time, weil das Generieren Zeit benötigt). Diese Schlüssel können Sie für `RSA` und `DSA` über die `ToXmlString`-Methode in XML-Form auslesen und z. B. in Dateien speichern. Die Schlüsselgröße (die bei asymmetrischen Algorithmen variabel ist) sollten Sie allerdings zuvor über die Eigenschaft `KeySize` bestimmen. Beachten Sie aber, dass größere Schlüssel zu einer langsameren Ver- und Entschlüsselung führen und dass nur bestimmte Schlüsselgrößen möglich sind (siehe Tabelle 32).

```
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);

// Instanz der Klasse für den Verschlüsselungs-Algorithmus erzeugen
AsymmetricAlgorithm asymmetricAlgorithm = RSA.Create();

// Die Schlüsselgröße bestimmen
asymmetricAlgorithm.KeySize = 2048;

// XML-String für den öffentlichen Schlüssel erzeugen
string publicKeyXml = asymmetricAlgorithm.ToXmlString(false);

// XML-String für den privaten und öffentlichen Schlüssel erzeugen
string privateAndPublicKeyXml = asymmetricAlgorithm.ToXmlString(true);

// Speichern als XML-Dateien
string privateAndPublicKeyFileName = Path.Combine(appPath,
   "PrivateAndPublicKey.xml");
string publicKeyFileName = Path.Combine(appPath, "PublicKey.xml");
File.WriteAllText(privateAndPublicKeyFileName, privateAndPublicKeyXml);
File.WriteAllText(publicKeyFileName, publicKeyXml);
```

Listing 589: Erzeugen von Schlüsseldateien

Zum Verschlüsseln von Daten über RSA benötigen Sie eine Referenz vom Typ der konkreten Klasse, da die abstrakten Klassen keine Methoden zum Ver- und Entschlüsseln zur Verfügung stellen.

Zum Verschlüsseln über RSA erzeugen Sie also eine Instanz der `RSACryptoServiceProvider`-Klasse. Den öffentlichen Schlüssel des Empfängers (den Sie in Form einer entsprechenden XML-Datei vorliegen haben) weisen Sie über die `FromXmlString`-Methode zu. Verschlüsseln können Sie dann über die `Encrypt`-Methode, aber:

Achtung

`Encrypt` erwartet neben den zu verschlüsselnden Daten am zweiten Argument einen booleschen Wert, der aussagt, ob die Verschlüsselung mit »OAEP-Padding« (auch: PKCS#1 Version 2) ausgeführt wird oder nicht (dann wird PKCS#1 Version 1.5 verwendet). OAEP-Padding kann nur auf Windows-Systemen ab XP verwendet werden.

RSA ist auf eine Maximallänge der zu verschlüsselnden Daten eingeschränkt. Versuchen Sie, zu große Daten zu verschlüsseln, erhalten Sie eine `CryptographicException` mit der wenig sagenden Meldung »Ungültige Länge«.

Die Maximallänge wird vom verwendeten Padding bestimmt, aber auch von dem verwendeten »Modulus« und der Länge eines Hashcode. Beim OAEP-Padding berechnet sich laut der Dokumentation die Maximallänge folgendermaßen:

Modulusgröße -2 -2 * *Hashgröße*

Beim PKCS#1-Version-1.5-Padding berechnet sich die Maximalgröße so:

Modulusgröße -11

Der »Modulus« einer RSA-Verschlüsselung ist ein nach dem Modulo-Verfahren berechneter Wert für den verwendeten Schlüssel. Dieses Verfahren wird bei Wikipedia erläutert: *en.wikipedia.org/wiki/Modular_arithmetic*.

Die Modulogröße können Sie über die RSA-Parameter ermitteln, die Sie in Form einer `RSAParameters`-Instanz über die `ExportParameters`-Methode exportieren können. Die Eigenschaft `Modulus` verwaltet ein Byte-Array mit den Modulo-Werten. In meinem Fall war die Größe dieses Arrays 128 Byte. Die RSA-Parameter können Sie auch neu definieren oder verändern und über die `ImportParameters`-Methode importieren. Allerdings hatte ich damit bisher keinen Erfolg ...

Wie Sie allerdings die *Hashgröße* ermitteln oder definieren können, ist nicht dokumentiert.

RSA kann also nur Daten bis zu einer Maximalgröße verschlüsseln. Durch reines Ausprobieren habe ich ermittelt, dass Sie mit dem OAEP-Padding maximal 87 Byte und mit dem PKCS#1-Version-1.5-Padding maximal 117 Byte verschlüsseln können. Dies habe ich unter Windows Vista und unter XP nachvollzogen. Ob das aber grundsätzlich so ist, ist mir unklar.

Weil RSA in der Praxis hauptsächlich eingesetzt wird, den Schlüssel einer symmetrischen Verschlüsselung zu verschlüsseln, ist diese Einschränkung nicht weiter schlimm. Die auch mit dem OAEP-Padding mögliche Schlüsselgröße von 696 Bit wird bei der symmetrischen Verschlüsselung nicht benötigt.

Listing 590 zeigt das Verschlüsseln eines Byte-Array, das als Schlüssel für die symmetrische Verschlüsselung verwendet werden soll. Das Ergebnis konvertiere ich allerdings in einen String, um diesen z. B. neben den einer E-Mail anzufügen, und aus diesem wieder in ein Byte-Array:

```
// Die zu verschlüsselnden Daten als Byte-Array definieren
byte[] symmetricEncryptionKey = { 115, 104, 64, 108, 104, 101, 105, 64,
   114, 104, 105, 101, 119, 110, 106, 107, 101, 104, 106, 101, 64, 119,
   104, 101, 104, 107, 100, 64, 115, 102, 43, 64, };
// RSA-Instanz erzeugen und den öffentlichen Schlüssel zuweisen
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
rsa.FromXmlString(File.ReadAllText(publicKeyFileName));

// Mit OAEP-Padding verschlüsseln
byte[] encryptedData = rsa.Encrypt(symmetricEncryptionKey, true);

// Die verschlüsselten Daten als ISO-8859-1-String darstellen
string encryptedString = Encoding.GetEncoding(
   "ISO-8859-1").GetString(encryptedData);
```

Listing 590: Asymmetrisches Verschlüsseln über den öffentlichen Schlüssel mit RSA

Entschlüsseln mit RSA

Zum Entschlüsseln von Daten müssen Sie über die `FromXmlString` das XML-Dokument zuweisen, das neben dem öffentlichen auch den privaten Schlüssel enthält. Entschlüsseln können Sie dann über die `Decrypt`-Methode. Dabei müssen Sie – neben dem korrekten privaten Schlüssel – dasselbe Padding verwenden.

```
// Die verschlüsselten Daten aus dem String ermitteln
encryptedData = Encoding.GetEncoding(
   "ISO-8859-1").GetBytes(encryptedString);

// RSA-Instanz erzeugen und mit dem XML-Schlüssel initialisieren
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
rsa.FromXmlString(File.ReadAllText(privateAndPublicKeyFileName));

// Entschlüsseln
byte[] decryptedData = rsa.Decrypt(encryptedData, true);
```

Listing 591: Asymmetrisches Entschlüsseln von Daten, die als String vorliegen

Falls Sie versuchen, lediglich über den öffentlichen Schlüssel zu entschlüsseln, resultiert dies in einer `CryptographicException`.

311 Sicherer Schlüsselaustausch mit ECDH

Der Elliptic Curve Diffie-Hellman-Algorithmus (ECDH) ist sicherer als RSA, weil er ein sehr spezielles Verfahren einsetzt, das auf der Potenzierung der zu verschlüsselnden Daten mit großen Exponenten basiert. Eine solche Verschlüsselung ist nur sehr schwer zu knacken. ECDH ist zurzeit der aktuelle Standard für die Verschlüsselung von Schlüsseln und wird von der *National Security Agency* als der beste Weg beschrieben, eine private Kommunikation zu sichern (*www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm*).

Die Grundidee von ECDH entspricht der von RSA: ECDH ist lediglich zum Austausch von Schlüsseln zwischen zwei Kommunikationspartnern vorgesehen, die für eine symmetrische Verschlüsselung von parallel (oder später) übertragenen Daten verwendet werden.

ECDH arbeitet aber andersherum: ECDH verschlüsselt nicht einen vorhandenen Schlüssel für die symmetrische Verschlüsselung. ECDH *erzeugt* einen Schlüssel, der für die symmetrische Verschlüsselung eingesetzt wird. Zwischen den Kommunikationspartnern wird nur der öffentliche Schlüssel ausgetauscht. Kommunikationspartner *A* erzeugt eine Instanz der `ECDiffieHellmanCng`-Klasse. Er ruft die `DeriveKeyMaterialMethode` auf, der er den öffentlichen Schlüssel des Kommunikationspartners *B* übergibt. Diese Methode gibt einen privaten Schlüssel zurück, der für die symmetrische Verschlüsselung der eigentlichen Daten verwendet werden kann.

B macht genau dasselbe wie *A*, nur eben mit dem öffentlichen Schlüssel von *A*.

Beide müssen sich zuvor noch auf eine Schlüssel-Extraktions-Funktion einigen. Diese wird in der Eigenschaften `KeyDerivationFunction` eingestellt. `KeyDerivationFunction` erwartet einen Wert der `ECDiffieHellmanKeyDerivationFunction`-Aufzählung. Die folgenden Werte sind definiert:

- `Hash`: Zur Generierung des Schlüssels wird ein Hash-Algorithmus verwendet. Die `HashAlgorithm`-Eigenschaft spezifiziert diesen über eine `CngAlgorithm`-Instanz, die über statische Eigenschaften dieser Klasse erzeugt werden.
- `Hmac`: Zur Generierung des Schlüssels wird der HMAC-Algorithmus verwendet. Die Eigenschaft `HmacKey` gibt den zu verwendenden Schlüssel an. Alternativ kann `UseSecretAgreementAsHmacKey` auf `true` gesetzt werden.
- `Tls`: Zum Generieren des Schlüssels wird das TLS (Transport Layer Security)-Protokoll verwendet. Dazu müssen Sie die Eigenschaften `Seed` und `Label` festlegen.

Die auf beiden Seiten erzeugten privaten Schlüssel sind dann gleich.

Das folgende Beispiel demonstriert dies, indem es beide Seiten simuliert. Es nutzt die Tatsache, dass die Klasse `ECDiffieHellmanCng` einen öffentlichen Schlüssel per Zufall generiert, der aus der Eigenschaft `PublicKey` in Form einer `ECDiffieHellmanPublicKey`-Instanz ausgelesen werden kann. Diese ist serialisierbar und kann deswegen z. B. über WCF ausgetauscht werden.

```
// Kommunikationspartner A erzeugt einen öffentlichen Schlüssel
ECDiffieHellmanCng ecDiffieHellmanA = new ECDiffieHellmanCng();
ECDiffieHellmanPublicKey publicKeyA = ecDiffieHellmanA.PublicKey;
```

Listing 592: Simulation der Erzeugung eines privaten Schlüssels auf zwei Seiten einer Kommunikation mit ECDH

```
// Kommunikationspartner B erzeugt ebenfalls einen öffentlichen Schlüssel
ECDiffieHellmanCng ecDiffieHellmanB = new ECDiffieHellmanCng();
ECDiffieHellmanPublicKey publicKeyB = ecDiffieHellmanB.PublicKey;

// A sendet B seinen öffentlichen Schlüssel
// B sendet A seinen öffentlichen Schlüssel

// A erzeugt einen privaten Schlüssel mit dem öffentlichen
// Schlüssel von B
ecDiffieHellmanA.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;
ecDiffieHellmanA.HashAlgorithm = CngAlgorithm.Sha512;
byte[] privateKeyA = ecDiffieHellmanA.DeriveKeyMaterial(publicKeyB);

// B erzeugt einen privaten Schlüssel mit dem öffentlichen
// Schlüssel von A
ecDiffieHellmanB.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;
ecDiffieHellmanB.HashAlgorithm = CngAlgorithm.Sha512;
byte[] privateKeyB = ecDiffieHellmanB.DeriveKeyMaterial(publicKeyA);
```

Listing 592: Simulation der Erzeugung eines privaten Schlüssels auf zwei Seiten einer Kommunikation mit ECDH (Forts.)

Abbildung 171 zeigt das Beispielprogramm zu diesem Rezept, das zusätzlich beide erzeugten Schlüssel ausgibt.

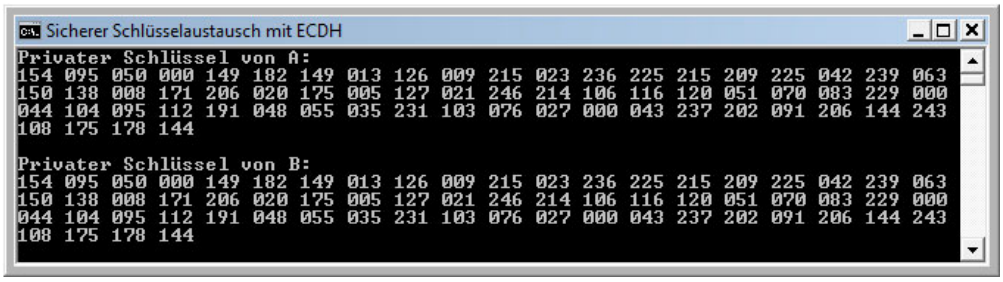


Abbildung 171: Das Beispielprogramm zum ECDH-Rezept zeigt beide Schlüssel an

312 Daten mit Hashing-Verfahren verschlüsseln

Daten müssen in einigen Fällen so verschlüsselt werden, dass diese nicht mehr entschlüsselt werden können und in einer verkürzten Form dargestellt werden. Dazu werden so genannte Hashcodes verwendet, die über öffentlich bekannte Hashing-Verfahren errechnet werden. Ein Hashcode repräsentiert die Daten in einer verkleinerten Form mit einer in der Regel festgelegten Länge (z. B. 128 oder 256 Bit). Aus einem Hashcode können unter keinen Umständen die eigentlichen Daten zurückgerechnet werden. Wird auf dieselben Daten derselbe Hashing-Algorithmus mit derselben Hashcode-Länge angewendet, resultiert aber immer derselbe Hashcode. Die Wahrscheinlichkeit, dass unterschiedliche Daten zu demselben Hashcode führen, ist äußerst gering.

Eine Anwendung für Hashcodes ist die Speicherung von Passwörtern, die von Anwendern zur Authentifizierung eingegeben werden sollen, in einer Konfigurationsdatei, Datenbank o. Ä. Gespeichert werden nicht die Passwörter, sondern deren Hashcodes. Im Login-Formular wird zur Authentifizierung das vom Anwender eingegebene Passwort in den korrespondierenden Hashcode umgerechnet und mit dem gespeicherten verglichen. Sind beide Hashcodes identisch, ist das Passwort korrekt. Natürlich können Sie für einen solchen Zweck auch einen Verschlüsselungs-Algorithmus verwenden. Hashcodes sind aber einfacher zu errechnen und bieten die Sicherheit, dass die Daten nicht entschlüsselt werden können.

Eine andere Anwendung für Hashcodes sind komplexe Verschlüsselungsverfahren, bei denen aus den zu verschlüsselnden Daten ein Hashcode berechnet wird, der den verschlüsselten Daten beigelegt wird. Der Empfänger kann aus den entschlüsselten Daten wieder einen Hashcode berechnen und diesen mit dem mitgelieferten vergleichen. Sind beide Hashcodes gleich, wurden die Daten mit einer recht hohen Wahrscheinlichkeit nicht verändert.

Hashcodes erzeugen Sie über spezielle Hashing-Klassen, die das .NET Framework im Namensraum `System.Security.Cryptography` zur Verfügung stellt. Diese Klassen implementieren die verschiedenen von .NET unterstützten Algorithmen und ermöglichen die Erzeugung von Hashcodes aus Byte-Arrays oder Streams.

Alle Hashing-Klassen sind (mehr oder weniger direkt) von der abstrakten Klasse `HashAlgorithm` abgeleitet. Für die Hashing-Algorithmen ohne Schlüssel reicht eine Referenz von diesem Typ aus. Die Hashing-Algorithmen mit Schlüssel sind von `KeyedHashAlgorithm` abgeleitet. `KeyedHashAlgorithm` definiert die zusätzliche Eigenschaft `Key`.

Zudem existieren weitere abstrakte Basisklassen wie z. B. `SHA1` oder `MD5`, die keine Implementierung beinhalten. Die Namen der davon abgeleiteten Klassen enden teilweise auf »Managed«, was darauf hinweist, dass die Implementierung in verwaltetem Code existiert (z. B. `SHA1Managed`). Namen mit der Endung »ServiceProvider« (z. B. bei der Klasse `SHA1CryptoServiceProvider`) weisen darauf hin, dass die Klassen lediglich Hüllen für entsprechende Klassen des Microsoft-Crypto-API sind. Daneben existieren auch Implementierungen mit der Endung »Cng« (z. B. `SHA1Cng`). Diese Endung steht für »Cryptography Next Generation« (CNG). CNG ist ein neues Kryptographie-Framework unter Windows Vista, mit dem Vorteil, dass der Kryptographie-Provider (z. B. bei einem Vista-Update) ausgetauscht werden kann, ohne dass ein Programmcode davon negativ betroffen ist (angeblich ...).

`HashAlgorithm` und alle abgeleiteten Klassen stellen die statische Methode `Create` zur Verfügung, die in den spezialisierteren Klassen überschrieben wird. Dieser Methode übergeben Sie einen String, der den zu verwendenden Algorithmus beschreibt. Die möglichen Strings sind in der Dokumentation der `Create`-Methode von `HashAlgorithm` dokumentiert. Über `HashAlgorithm.Create("SHA512")` erzeugen Sie z. B. eine Instanz für den SHA-Algorithmus mit einer Schlüssellänge von 512 Bit.

Für die spezialisierten Klassen, die die `Create`-Methode von `HashAlgorithm` erben, können Sie nur noch Strings übergeben, die zu der Klasse passende Algorithmen beschreibt. In anderen Fällen erhalten Sie eine `InvalidCastException`.

Die abstrakten Methoden auf der letzten Ebene fügen zudem eine `Create`-Methode ohne Argument hinzu. Über diese Methode erzeugen Sie laut der (wenig helfenden) Dokumentation eine Instanz der »Standardimplementierung« des jeweiligen Verfahrens. Nach meinen Erfahrungen ist dies die »ServiceProvider«-Implementierung, falls eine vorhanden ist.

Schließlich überschreiben die konkreten Klassen die `Create`-Methoden. Damit können Sie dann auch eine Instanz dieser speziellen Implementierung des Algorithmus erzeugen. Unsinnige Aufrufe wie z. B. `MD5Cng.Create("SHA1")` resultieren in einer `InvalidCastException`. Die konkreten Klassen lassen an dem String-Argument `Create`-Methode nur noch ihren speziellen String zu (was die Anwendung dieser Variante der `Create`-Methode dann sinnlos macht).

Tabelle 33 beschreibt die jeweiligen abstrakten Basisklassen und deren Implementierungen. Diese Tabelle beschreibt gleich noch die maximale Größe des Schlüssels (nach eigenen Versuchen), damit in der Praxis keine Probleme auftreten ☺.

Basisklasse	Implementierungen	Beschreibung	Maximale Schlüsselgröße
`MD5`	`MD5Crypto-ServiceProvider` `MD5Cng`	MD5 (Message Digest 5) mit 128 Bit Länge	Kein Schlüssel
Direkt von `Hash-Algorithm` abgeleitet	`RIPEMD160`	RIPEMD-Verfahren mit 160 Bit Länge	Kein Schlüssel
`SHA1`	`SHA1Managed` `SHA1Cng`	SHA1 (Secure Hash Algorithm 1) mit 160 Bit Länge	Kein Schlüssel
`SHA256`	`SHA256Managed` `SHA256Cng`	SHA1 mit 256 Bit Länge	Kein Schlüssel
`SHA384`	`SHA384Managed` `SHA384Cng`	SHA1 mit 384 Bit Länge	Kein Schlüssel
`SHA512`	`SHA512Managed` `SHA512Cng`	SHA1 mit 512 Bit Länge	Kein Schlüssel
`HMAC`	`HMACMD5`	MD5 mit 128 Bit Länge und Schlüssel	64 Byte
`HMAC`	`HMACRIPEMD160`	RIPEMD160 mit 160 Bit Länge und Schlüssel	64 Byte
`HMAC`	`HMACSHA1`	SHA1 mit 160 Bit und Schlüssel	64 Byte
`HMAC`	`HMACSHA256`	SHA256 mit 256 Bit und Schlüssel	64 Byte
`HMAC`	`HMACSHA384`	SHA384 mit 384 Bit und Schlüssel	128 Byte
`HMAC`	`HMACSHA512`	SHA512 mit 512 Bit und Schlüssel	128 Byte
Direkt von `KeyedHash-Algorithm` abgeleitet	`MACTripleDES`	TripleDES mit 64 Bit Länge und Schlüssel	24 Byte

Tabelle 33: Die .NET-Klassen zum Erzeugen von Hashcodes

Achtung

Laut der Dokumentation werden die CNG-Varianten unter Windows Vista, Windows XP SP2 und Windows Server 2003 unterstützt. Auf meinem Windows XP SP2 erhielt ich aber eine `PlatformNotSupportedException`. Das ist auch logisch, denn CNG ist ja schließlich ein neues Kryptographie-Framework unter *Vista*.

Die verschiedenen Implementierungen des SHA1-Algorithmus verwenden eine unterschiedliche Hashcode-Größe. Ein größerer Hashcode, dessen Berechnung ein wenig mehr Zeit kostet, bringt eine größere Sicherheit gegenüber Doppelgängern. Die Größe des Hashcodes können Sie übrigens aus der Eigenschaft `HashSize` auslesen.

Die Klassen, deren Name mit *HMAC* (Hash-based Message Authentication Code) beginnt, verwenden einen privaten Schlüssel, der mit den Daten vermischt wird. Das Ergebnis wird über das verwendete Hashing-Verfahren in einen Hashcode umgewandelt, der Hashcode wird wieder mit dem privaten Schlüssel vermengt und die Hashfunktion wird ein zweites Mal auf diese Datenmenge angewendet. Auf diese Weise erzeugte Hashcodes sind sehr sicher und werden üblicherweise für das Signieren von Nachrichten verwendet.

Die Klasse `MACTripleDES` arbeitet wahrscheinlich ähnlich (was aber nicht dokumentiert ist): *MAC* steht für *Message Authentication Code*.

Direktes Erzeugen eines Hashcodes

Die Verwendung der Hashing-Klassen ist sehr einfach: Sie erzeugen eine Instanz und rufen zur Erzeugung des Hashcodes die Methode `ComputeHash` auf. Dieser Methode können Sie ein Byte-Array oder einen Stream übergeben. Sie erhalten dann ein Byte-Array mit den Daten des Hashcodes zurück.

Bei den Algorithmen, die mit einem Schlüssel arbeiten (die HMAC-Klassen und die Klasse `MACTripleDES`), können Sie zudem vor der Erzeugung des Hashcodes in der Eigenschaft `Key` den Schlüssel definieren. Dieser Schlüssel wird bei jeder Erzeugung einer Instanz dieser Klassen per Zufall neu erzeugt. Für den Vergleich zweier Hashcodes müssen Sie den Schlüssel also zwischenspeichern und beim zweiten Ermitteln des Hashcodes setzen. Außerdem darf der Schlüssel eine vom Algorithmus bestimmte Maximalgröße nicht überschreiten (siehe Tabelle 33).

Das folgende Beispiel erzeugt einen Haschcode für eine Datei über das HMAC-SHA-512-Verfahren. Als Schlüssel (128 Bit) wird ein String verwendet (weil das praxisorientierter ist). Der Schlüssel wird über die Codierung ISO-8859-1 in ein Byte-Array umgerechnet, damit (bei 16 Zeichen) 16 Byte herauskommen und keine Daten verloren gehen. Schließlich wandelt das Beispiel noch den Hashcode in einen String um, wobei wieder die Codierung ISO-8859-1 verwendet wird um Probleme zu vermeiden:

```
// Datei in einen Stream lesen
string appPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string fileName = Path.Combine(appPath, "DontPanic.gif");
using (FileStream fs = new FileStream(fileName,
```

Listing 593: Typisches Erzeugen eines Hashcodes

```
   FileMode.Open, FileAccess.Read))
{
   // Einen Schlüssel für den Hashcode aus einem String berechnen
   Encoding encoding = Encoding.GetEncoding("ISO-8859-1");
   // 16 Zeichen = 128 Bit in ISO-8859-1
   string hashCodeKeyString = "k4odf#+ä@sd+ksxh";
   byte[] hashCodeKey = encoding.GetBytes(hashCodeKeyString);

   // Für die Daten einen Hashcode berechnen
   HMACSHA512 hashAlgorithm = new HMACSHA512(hashCodeKey);
   byte[] hashcode = hashAlgorithm.ComputeHash(fs);

   // Den Hashcode in einen String umwandeln
   string hashCodeString = encoding.GetString(hashcode);

   // Mit dem Hashcode arbeiten
   ...
}
```

Listing 593: Typisches Erzeugen eines Hashcodes (Forts.)

Eine Hilfsklasse

Um die Arbeit mit Hashcodes zu erleichtern und um die Übergabe des Schlüssels für die Algorithmen zu erleichtern, die einen solchen verwenden, habe ich die Klasse `Hasher` entwickelt. Die Grundidee dieser Klasse ist dieselbe wie die der Klasse `SymmetricEncryptor` aus dem Rezept 309. `Hasher` ermöglicht die einfache Verwendung aller unterstützten Hashing-Algorithmen. Da Hashcodes häufig als String gespeichert werden, geben die Varianten der Methode `ComputeHash`, die den Hashcode erzeugen, einfach einen String in der zu Unicode kompatiblen 8-Bit-Codierung ISO-8859-1 zurück.

Zum Kompilieren der `Hasher`-Klasse müssen Sie die Namensräume `System`, `System.IO`, `System.Text` und `System.Security.Cryptography` importieren.

Zunächst habe ich für die Definition des zu verwendenden Algorithmus die Aufzählung `HashAlgorithmKind` implementiert:

```
public enum HashAlgorithmKind
{
   MD5,
   MD5Cng,
   RIPEMD160,
   SHA1,
   SHA1Cng,
   SHA256,
   SHA256Cng,
   SHA384,
```

Listing 594: Aufzählung für die unterstützten Hashing-Algorithmen

```
    SHA384Cng,
    SHA512,
    SHA512Cng,
    HMACMD5,
    HMACRIPEMD160,
    HMACSHA1,
    HMACSHA256,
    HMACSHA384,
    HMACSHA512,
    MACTripleDES
}
```

Listing 594: Aufzählung für die unterstützten Hashing-Algorithmen (Forts.)

Die Klasse `Hasher` beginnt mit Feldern zur Speicherung der verwendeten `HashAlgorithm`-Instanz, der zum Codieren von Strings verwendeten Codierung und einer Eigenschaft, die im Konstruktor mit der Maximallänge des Schlüssels belegt wird. Am Konstruktor wird außerdem der zu verwendende Algorithmus übergeben und in der privaten Variablen `hashAlgorithm` gespeichert:

```
public class Hasher
{
    /* Verwaltet das Hash-Objekt */
    private HashAlgorithm hashAlgorithm;

    /* Die Codierung für das Konvertieren von Strings in Byte-Arrays
       und umgekehrt. Es muss sich dabei zwingend um eine 8-Bit-Codierung
       handeln */
    private Encoding stringEncoding = Encoding.GetEncoding("ISO-8859-1");

    /* Verwaltet die maximale Schlüssel-Länge */
    public int MaxKeyLength
    {
        private set;
        get;
    }

    /* Konstruktor */
    public Hasher(HashAlgorithmKind algorithm)
    {
        // Algorithmus definieren
        switch (algorithm)
        {
            case HashAlgorithmKind.MD5:
                this.hashAlgorithm = new MD5CryptoServiceProvider();
```

Listing 595: Start der Hasher-Klasse mit Feldern, einer Eigenschaft und dem Konstruktor

```
            break;

        case HashAlgorithmKind.MD5Cng:
            this.hashAlgorithm = new MD5Cng();
            break;

        case HashAlgorithmKind.RIPEMD160:
            this.hashAlgorithm = new RIPEMD160Managed();
            break;

        case HashAlgorithmKind.SHA1:
            this.hashAlgorithm = new SHA1Managed();
            break;

        case HashAlgorithmKind.SHA1Cng:
            this.hashAlgorithm = new SHA1Cng();
            break;

        case HashAlgorithmKind.SHA256:
            this.hashAlgorithm = new SHA256Managed();
            break;

        case HashAlgorithmKind.SHA256Cng:
            this.hashAlgorithm = new SHA256Cng();
            break;

        case HashAlgorithmKind.SHA384:
            this.hashAlgorithm = new SHA384Managed();
            break;

        case HashAlgorithmKind.SHA384Cng:
            this.hashAlgorithm = new SHA384Cng();
            break;

        case HashAlgorithmKind.SHA512:
            this.hashAlgorithm = new SHA512Managed();
            break;

        case HashAlgorithmKind.SHA512Cng:
            this.hashAlgorithm = new SHA512Cng();
            break;

        case HashAlgorithmKind.HMACMD5:
            this.hashAlgorithm = new HMACMD5();
            break;
```

Listing 595: Start der Hasher-Klasse mit Feldern, einer Eigenschaft und dem Konstruktor (Forts.)

```
            case HashAlgorithmKind.HMACRIPEMD160:
               this.hashAlgorithm = new HMACRIPEMD160();
               break;

            case HashAlgorithmKind.HMACSHA1:
               this.hashAlgorithm = new HMACSHA1();
               break;

            case HashAlgorithmKind.HMACSHA256:
               this.hashAlgorithm = new HMACSHA256();
               break;

            case HashAlgorithmKind.HMACSHA384:
               this.hashAlgorithm = new HMACSHA384();
               break;

            case HashAlgorithmKind.HMACSHA512:
               this.hashAlgorithm = new HMACSHA512();
               break;

            case HashAlgorithmKind.MACTripleDES:
               this.hashAlgorithm = new MACTripleDES();
               break;
         }

         // Die Maximallänge des Schlüssels ermitteln
         if (this.hashAlgorithm is KeyedHashAlgorithm)
         {
            this.MaxKeyLength =
               ((KeyedHashAlgorithm)this.hashAlgorithm).Key.Length;
         }
         else
         {
            this.MaxKeyLength = 0;
         }
      }
```

Listing 595: Start der Hasher-Klasse mit Feldern, einer Eigenschaft und dem Konstruktor (Forts.)

Damit für die Algorithmen, die Schlüssel verwenden (HMAC..., `MACTripleDES`), der Schlüssel definiert werden kann, besitzt die Klasse `Hasher` eine Eigenschaft `Key`, der Sie den Schlüssel als String übergeben können. Um sicherzustellen, dass der in einem Byte-Array verwaltete Schlüssel ausgelesen bzw. gesetzt werden kann, gibt die Eigenschaft `SupportsKey` zurück, ob ein Schlüssel unterstützt wird. Diese Eigenschaft liest dazu einfach die im Konstruktor gesetzte `MaxKeyLength`-Eigenschaft aus. `SupportsKey` wird in den Accessoren von `Key` abgefragt um im negativen Fall eine `NotSupportedException` zu werfen.

In den Accessoren der `Key`-Eigenschaft verwende ich als Codierung für den String die zu Unicode kompatible ISO-8859-1-Codierung (die über das Feld `stringEncoding` verwaltet wird). Die

Begründung für den Verzicht auf den kompletten Unicode-Zeichensatz liegt darin, dass die meisten Strings in unseren Breiten lediglich den 8-Bit-Bereich verwenden. Bei einer Unicode-Codierung wäre jedes zweite Byte des Schlüssels ggf. 0 (außer bei UTF-8, die eine dynamische Byteanzahl pro Zeichen verwendet), was diesen in der Qualität verschlechtern würde. Außerdem könnte die Länge des Schlüssels bei Codierungen wie UTF-8 nur sehr schwer am String erkannt werden, wenn Zeichen mit einem Wert größer als 255 im Schlüssel enthalten sind.

Der `set`-Accessor von `Key` überprüft den übergebenen Schlüssel deshalb darauf, ob dieser Zeichen mit einem Code größer 0x00FF speichert. Damit wird sichergestellt, dass keine Unicode-Zeichen übergeben werden, die den 8-Bit-Bereich überschreiten. Außerdem wird die Länge des Schlüssels mit der Maximallänge verglichen und im Fall der Überschreitung eine `NotSupportedException` geworfen.

```
/* Gibt zurück, ob der aktuell verwendete Algorithmus
   einen Schlüssel erlaubt */
public bool SupportsKey
{
   get
   {
      return (this.MaxKeyLength > 0);
   }
}

/* Verwaltet den Schlüssel für Algorithmen, die einen solchen benötigen */
public string Key
{
   get
   {
      // Überprüfen, ob der Algorithmus einen Schlüssel erlaubt
      if (this.SupportsKey)
      {
         // Schlüssel in einen String umwandeln und zurückgeben
         return this.stringEncoding.GetString(
            ((KeyedHashAlgorithm)this.hashAlgorithm).Key);
      }
      else
      {
         throw new NotSupportedException("Der aktuell verwendete " +
            "Hash-Algorithmus unterstützt keine Schlüssel");
      }
   }

   set
   {
```

Listing 596: Eigenschaften zur Abfrage, ob ein Schlüssel unterstützt wird, und zum Lesen und Schreiben des Schlüssels

```
            // Auf Unicode-Zeichen größer 0x00FF überprüfen
            for (int i = 0; i < value.Length; i++)
            {
               if ((int)value[i] > 255)
               {
                  throw new CryptographicException("Der übergebene " +
                     "Schlüssel enthält mindestens ein Unicode-Zeichen, " +
                     "das größer ist als 0x00FF (255): " + value[i] + " (" +
                     (int)value[i] + "). Unterstützt werden lediglich " +
                     "8-Bit-Unicode-Zeichen");
               }
            }

            // Überprüfen, ob der Algorithmus einen Schlüssel erlaubt
            if (this.SupportsKey)
            {
               // Schlüssel in ein Byte-Array überführen
               byte[] key = this.stringEncoding.GetBytes(value);

               // Überprüfen, ob die Schlüssellänge zum Algorithmus passt.
               if (key.Length <= this.MaxKeyLength)
               {
                  // Schlüssel setzen
                  ((KeyedHashAlgorithm)this.hashAlgorithm).Key = key;
               }
               else
               {
                  throw new NotSupportedException("Der übergebene Schlüssel " +
                     "ist mit " + key.Length + " Byte zu groß für den " +
                     "gesetzten Hash-Algorithmus. Dieser unterstützt nur " +
                     "maximal " + this.MaxKeyLength + " Byte große Schlüssel");
               }
            }
            else
            {
               throw new NotSupportedException("Der aktuell verwendete " +
                  "Hash-Algorithmus unterstützt keine Schlüssel");
            }
         }
      }
```

Listing 596: Eigenschaften zur Abfrage, ob ein Schlüssel unterstützt wird, und zum Lesen und Schreiben des Schlüssels (Forts.)

Nun folgen nur noch die wichtigen Methoden zum Erzeugen des Hashcodes. Die vier Varianten der Methode `ComputeHash` erwarten ein Byte-Array, einen Stream oder einen String als Quelle, erzeugen daraus einen Hashcode und geben diesen als ISO-8859-1-String zurück.

```
   /* Erzeugt einen Hash aus einem Byte-Array */
   public string ComputeHash(byte[] data)
   {
      return this.stringEncoding.GetString(
         this.hashAlgorithm.ComputeHash(data));
   }

   /* Erzeugt einen Hash aus einem Byte-Array */
   public string ComputeHash(byte[] data, int offset, int count)
   {
      return this.stringEncoding.GetString(
         this.hashAlgorithm.ComputeHash(data, offset, count));
   }

   /* Erzeugt einen Hash aus den Daten eines Stream */
   public string ComputeHash(Stream inputStream)
   {
      return this.stringEncoding.GetString(
         this.hashAlgorithm.ComputeHash(inputStream));
   }

   /* Erzeugt einen Hash für einen String */
   public string ComputeHash(string inputString)
   {
      // Byte-Array aus dem String erzeugen und damit den Hashcode erzeugen
      byte[] buffer = Encoding.Unicode.GetBytes(inputString);
      return this.stringEncoding.GetString(
         this.hashAlgorithm.ComputeHash(buffer));
   }
}
```

Listing 597: Methoden zur Erzeugung eines Hashcodes

Die `Hasher`-Klasse ist zwar etwas aufwändiger geworden, als ich ursprünglich vermutete. Die Anwendung dieser Klasse ist dafür aber sehr einfach. Das folgende Beispiel erzeugt aus einem String über das HMACSHA1-Verfahren (das mit einem Schlüssel arbeitet) einen Hashcode:

```
Hasher hasher = null;
try
{
   // Hasher erzeugen und Schlüssel übergeben
   hasher = new Hasher(HashAlgorithmKind.HMACSHA1);
   hasher.Key = "löke~08325~ybnk~y9732~35298~46t~";
}
catch (Exception ex)
```

Listing 598: Erzeugen eines Hashcodes für einen String

```
{
   Console.WriteLine(ex.Message);
   return;
}

Console.WriteLine("Hashing nach dem HMACSHA1-Verfahren:");

// Den Hashcode für einen String ermitteln
string source = "Das ist ein Teststring zur Ermittlung eines Hashcodes";
string hashCode = hasher.ComputeHash(source);
Console.WriteLine("Quellstring: {0}", source);
Console.WriteLine("Hashcode: {0}", hashCode);
```

Listing 598: Erzeugen eines Hashcodes für einen String (Forts.)

Sie können aber genauso gut einen Hashcode für eine Datei erzeugen, indem Sie `ComputeHash` ein `FileStream`-Objekt übergeben oder ein Byte-Array als Quelle verwenden.

Multimedia

313 Wave-Dateien abspielen

Zum Abspielen von Wave-Dateien (mit der Endung *.wav*) können Sie neben verschiedenen externen Techniken wie WMI und DirectX, die ich in den folgenden Rezepten beschreibe, auch die Klasse `SoundPlayer` aus dem Namensraum `System.Media` verwenden.

> **Hinweis**
>
> Leider unterstützt diese Klasse lediglich das Wave-Format. Zum Abspielen von anderen Formaten wie z. B. MP3 müssen Sie externe Techniken wie MCI (siehe Rezept 314) oder DirectX (nicht in diesem Buch behandelt) einsetzen.

Beim Erzeugen einer Instanz der `SoundPlayer`-Klasse übergeben Sie den Dateinamen der Wave-Datei oder einen Stream, der die Wave-Daten enthält. Dann rufen Sie die Methoden `Play`, `PlaySync` oder `PlayLooping` auf. `Play` spielt die Datei in einem separaten Thread ab. Der Thread, in dem diese Methode aufgerufen wird, wird also während des Abspielens nicht blockiert. `PlaySync` spielt die Sounddatei in dem Thread ab, der die Methode aufgerufen hat. Der Thread wird also so lange blockiert, bis das Abspielen beendet ist. Wenn Sie eine Wave-Datei mit `PlayLooping` abspielen, wird diese in einem neuen Thread so lange wiederholt abgespielt, bis Sie die `Stop`-Methode aufrufen. `Stop` können Sie natürlich auch aufrufen um das Abspielen einer mit `Play` einmalig gespielten Wave-Datei zu beenden.

Das folgende Beispiel spielt die Wave-Datei *pavel_und_bronko.wav*, die im Anwendungsordner erwartet wird, in einer Endlosschleife ab, die über eine Betätigung der Return-Taste in der Konsolenanwendung beendet werden kann. Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.IO`, `System.Media` und `System.Reflection` einbinden.

```
// Dateiname zusammensetzen
string fileName = Path.Combine(Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location), "pavel_und_bronko.wav");

// SoundPlayer erzeugen
SoundPlayer soundPlayer = new SoundPlayer(fileName);
try
{
   // Wave-Datei wiederholend (und gleichzeitig asynchron) abspielen
   soundPlayer.PlayLooping();
}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

Listing 599: Wiederholtes, asynchrones Abspielen einer Wave-Datei

```
Console.WriteLine("Stoppen Sie das Abspielen mit Return");
Console.ReadLine();

// Das Abspielen stoppen
soundPlayer.Stop();
```

Listing 599: Wiederholtes, asynchrones Abspielen einer Wave-Datei (Forts.)

Hinweis

Unter Windows 98 (und wahrscheinlich auch unter Windows Me) kann die Klasse `SoundPlayer` nicht in einer Konsolenanwendung ausgeführt werden. Das liegt daran, dass die Konsole unter Windows 98 im 16-Bit-Modus läuft, während die verwendete API-Funktion `PlaySound` eine 32-Bit-Funktion ist.

314 Multimedia-Dateien in einer Windows.Forms-Anwendung über MCI abspielen

Das relativ alte MCI (Media Control Interface) ermöglicht den Zugriff auf alle Multimedia-Dienste, die vom Betriebssystem zur Verfügung gestellt werden. MCI ist in Form von verschiedenen API-Funktionen in der Datei *winmm.dll* bereits seit Windows 3.1 Teil des Betriebssystems. In einer `Windows.Forms`-Anwendung bietet MCI (in Form meiner `Mci`-Klasse) eine gute Möglichkeit, Multimediadateien abzuspielen (in einer WPF-Anwendung steht dazu das `MediaElement`-Steuerelement zur Verfügung, siehe Rezept 315).

An Stelle von MCI können Sie in einer `Windows.Forms`-Anwendung auch DirectX für das Abspielen von Multimedia-Dateien verwenden. MCI besitzt aber den Vorteil, dass Sie davon ausgehen können, dass eigentlich jeder Computer, auf dem das .NET Framework (in der Windows-Version) installiert ist, über die notwendigen MCI-Treiber verfügt. DirectX muss hingegen ggf. separat installiert werden.

MCI arbeitet mit Treibern, die den Zugriff auf spezielle Multimedia-Geräte (bzw. -Programme) so kapseln, dass immer dieselben MCI-Funktionen zur Steuerung der Geräte verwendet werden können. Für spezielle Multimedia-Formate wie z. B. das Apple-QuickTime-Format können Sie spezielle MCI-Treiber beziehen (meist reicht dazu eine Google-Suche aus). Oft sind MCI-Treiber aber auch in der Installation eines Multimedia-Programms (wie z. B. bei QuickTime) enthalten. Die Microsoft-Treiber werden hauptsächlich vom Windows Media Player installiert. Ältere Betriebssysteme besitzen natürlich per Voreinstellung nur ältere Treiber und können mit Multimediadateien neuerer Generation nichts anfangen. Windows 98 ist allerdings nach einer frischen Installation (mit dem Internet Explorer 6 und dem .NET Framework 1.0) bereits in der Lage die wichtigsten Formate (Wave, MP3, AVI, MPEG) abzuspielen.

Falls eine Aktualisierung der Treiber notwendig sein sollte, reicht wahrscheinlich die Installation der neuesten Version des Media Players aus. Ein separater Download der Microsoft-MCI-Treiber scheint auf der Microsoft-Website leider nicht verfügbar zu sein.

MCI-Geräte können Sie im Wesentlichen über zwei Funktionen steuern. Der Funktion `mciSendCommand` übergeben Sie dazu eine Struktur mit den Befehlsparametern. Diese Variante ist etwas umständlich. Die einfachere Variante über die Funktion `mciSendString` arbeitet mit einem

Befehls-String und ist in der Anwendung wesentlich einfacher. Zur Arbeit mit MCI benötigen Sie dann nur noch die Funktion `mciGetErrorString` zum Auslesen aufgetretener Fehler und etwas Wissen um die Befehls-Strings. Der Vorgang bei der Arbeit mit MCI-Geräten ist immer derselbe: Das Gerät wird zuerst geöffnet, dann können Sie Befehle wie z. B. `Play` zum Abspielen oder `Stop` zum Stoppen aufrufen, zum Schluss wird das Gerät wieder geschlossen.

Problematisch ist dabei lediglich das Herausfinden der korrekten MCI-Befehlsstrings. Diese sind in der MCI-Dokumentation recht gut in Zusammenhang mit der `mciSendString`-Funktion beschrieben (über den Link MULTIMEDIA COMMAND STRINGS). Die MCI-Dokumentation finden Sie in der Hilfe, indem Sie nach »"MCI Reference"« suchen (inkl. der Anführungszeichen). Lassen Sie sich vom Namen des vorletzten Ordners nicht verwirren: Moderne Systeme liefern auch MCI-Treiber für Videodateien mit. Im Internet finden Sie die MCI-Referenz am besten, indem Sie bei *msdn.microsoft.com* nach »MCI« suchen. Die derzeitige Adresse der MCI-Referenz ist *msdn.microsoft.com/en-us/library/ms710984(VS.85).aspx*.

So, das reicht an Theorie. Nun folgt die Klasse `Mci`, die Ihnen die Arbeit mit MCI sehr erleichtert. Diese Klasse implementiert zwar lange nicht alle Möglichkeiten, die MCI bietet. Sie können aber über `Mci`-Instanzen Audio- und Video-Dateien abspielen. Trotzdem ist die Klasse bereits recht komplex. Aber die Arbeit (des Kopierens der Klasse aus dem Repository ...) lohnt sich: Sie erhalten damit eine sehr einfach anzuwendende Möglichkeit, Sound- und Videodateien abzuspielen.

Zum Kompilieren der Klasse müssen Sie die Assembly *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.IO`, `System.Text`, `System.Runtime.InteropServices` und `System.Windows.Forms` importieren.

Zur Implementierung habe ich zunächst eine Klasse `MciException` für eigene Ausnahmen deklariert. `MciException` erwartet die Fehlermeldung im Konstruktor.

```
public class MciException : Exception
{
   public MciException(string message)
   {
   }
}
```

Listing 600: Klasse für eigene Ausnahmen

Die Klasse `Mci` ist von `IDisposable` abgeleitet, damit der Aufruf der `Dispose`-Methode, die später implementiert wird, möglich ist. Am Anfang der Klasse habe ich die verwendeten API-Funktionen deklariert:

```
public class Mci : IDisposable
{
   /* Deklaration der MCI-Funktionen, -Konstanten und -Strukturen */
   [DllImport("winmm.dll", CharSet = CharSet.Auto)]
```

Listing 601: Beginn der Klasse MCI mit der Deklaration der verwendeten API-Funktionen

```
private static extern int mciSendString(string lpstrCommand,
   StringBuilder lpstrReturnString, int uReturnLength, IntPtr
   hwndCallback);

[DllImport("winmm.dll", CharSet = CharSet.Auto)]
private static extern int mciGetErrorString(int dwError, StringBuilder
   lpstrBuffer, int uLength);
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
private static extern int GetShortPathName(string lpszLongPath,
   StringBuilder lpszShortPath, int cchBuffer);
```

Listing 601: Beginn der Klasse MCI mit der Deklaration der verwendeten API-Funktionen (Forts.)

MCI-Geräte werden über eine beim Öffnen automatisch vergebene Geräte-Id oder einen Alias angesprochen. Ich verwende einen Alias, der im Fall der `Mci`-Klasse einfach ein GUID-Wert ist. Da dieser Alias in verschiedenen Methoden benötigt wird, speichert `Mci` diesen in dem privaten Feld `alias`.

Eine schreibgeschützte Eigenschaft `IsOpen` macht eine Aussage darüber, ob das MCI-Gerät geöffnet ist, und wird von den Methoden zum Öffnen und Schließen gesetzt.

Zur Vereinfachung der Ermittlung von MCI-Fehlern habe ich dann die Methode `GetMciError` geschrieben. Diese Methode ruft die Funktion `mciGetErrorString` mit dem übergebenen Fehlercode auf und gibt den ermittelten String zurück. `mciGetErrorString` gibt, wie viele API-Funktionen, die mit Strings arbeiten, die Anzahl der gelesenen Zeichen zurück. Bei einer Rückgabe von 0 (z. B. bei der Übergabe eines unbekannten Fehlercodes) gibt `GetMciError` lediglich eine generische Fehlermeldung zurück:

```
/* Feld für den MCI-Alias */
private string alias;

private bool isOpen = false;
/* Gibt an, ob gerade ein MCI-Gerät geöffnet ist */
public bool IsOpen
{
   get
   {
      return this.isOpen;
   }
}

/* Liefert eine Fehlermeldung zu einem gegebenen Fehlercode */
private string GetMciError(int errorCode)
{
   StringBuilder errorMessage = new StringBuilder(255);
   if (mciGetErrorString(errorCode, errorMessage,
```

Listing 602: Privates Feld, Eigenschaft und eine Methode zur Ermittlung eines MCI-Fehlers

```
            errorMessage.Capacity) == 0)
        {
            return "MCI-Fehler " + errorCode;
        }
        else
        {
            return errorMessage.ToString();
        }
    }
```

Listing 602: Privates Feld, Eigenschaft und eine Methode zur Ermittlung eines MCI-Fehlers (Forts.)

Nun folgen Methoden zum Öffnen eines MCI-Geräts (automatisch über den Dateinamen). Da MCI auch das Abspielen von Videodateien auf einem gegebenen Windows-Fenster (einem Formular, einem `PictureBox`-Steuerelement etc.) erlaubt, habe ich zwei Varianten der `Open`-Methode entwickelt. Die erste, die lediglich den Dateinamen der Multimedia-Datei übergeben bekommt, ruft dabei einfach die zweite auf. Diese zweite Variante, die neben dem Dateinamen eine `Control`-Referenz erwartet (womit Sie alle .NET-Steuerelemente und Formulare übergeben können), schließt zunächst ein eventuell zuvor geöffnetes Gerät, um Probleme beim gleichzeitigen Abspielen von mehreren Multimediadateien zu vermeiden. Danach erzeugt die Methode einen eindeutigen Alias für das MCI-Gerät. Sie nutzt dazu einfach die `NewGuid`-Methode der `Guid`-Klasse, die eine weltweit eindeutige GUID erzeugt. Um die nicht sinnvollen Bindestriche aus der String-Repräsentation des GUID-Werts zu entfernen, ruft `Open` die `ToString`-Methode der erzeugten `Guid`-Instanz mit dem Formatzeichen »N« (numerische Formatierung) auf.

Dann überprüft `Open`, ob die angegebene Datei existiert, und generiert im Fehlerfall eine Ausnahme vom Typ `FileNotFoundException`. Damit wird verhindert, dass MCI beim Öffnen einer nicht vorhandenen Datei einen Fehler erzeugt und dieser in einer `MciException` resultiert.

Der im `Open`-Befehl später übergebene Dateiname kann in Anführungszeichen gesetzt werden, damit MCI den Befehl korrekt interpretieren kann (MCI trennt die Befehlsparameter durch Leerzeichen). Nach Aussagen in einer Newsgroup hat MCI aber Probleme mit sehr langen Dateinamen, was ich unter Windows XP allerdings nicht nachvollziehen konnte. Zur Sicherheit setzt `Open` den langen Dateinamen über `GetShortPathName` in einen kurzen um und verzichtet im MCI-String auf die Anführungszeichen.

Dann wird der MCI-String zum Öffnen zusammengesetzt. Dieser sieht in unserem Fall beispielsweise zum Abspielen einer MP3-Datei folgendermaßen aus, wenn kein Besitzer-Fenster angegeben ist:

```
"open C:\Patience.mp3 type mpegvideo alias 9343e55ade624346bb9a0729425d0bb4"
```

Der Parameter `type` gibt den zu verwendenden Gerätetyp an. Laut der Dokumentation können Sie nur spezifische Gerätenamen wie z. B. `cdaudio` (CD-ROM), `waveaudio` (Wave-Dateien) oder `digitalvideo` (Videodateien) einsetzen. Etwas problematisch ist dabei die notwendige Unterscheidung der verschiedenen Typen.

Der undokumentierte Typ `mpegvideo` (der weder in der Dokumentation noch auf der Microsoft Website zu finden ist) spricht hingegen den MPEGVideo-Treiber (*mciqtz.drv*) an, der in der

Lage ist, alle wichtigen Audio- und Videoformate abzuspielen. Und das erleichtert die Arbeit ungemein.

Falls am zweiten Argument eine `Control`-Referenz übergeben wurde, hängt `Open` noch den Zusatz `parent` *Handle* `style child` mit dem Handle des Objekts an den MCI-Befehlsstring an, damit ein Video auf der Fläche des Controls ausgegeben wird.

Mit dem korrekt zusammengesetzten MCI-String ruft `Open` dann die `mciSendString`-Funktion auf. Da keine Daten zurückerwartet werden (einige MCI-Befehle lesen Statusdaten oder allgemeine Informationen zum Gerät aus), wird am zweiten Argument (das eine `StringBuilder`-Instanz erwartet) `null` und am dritten 0 übergeben. Das letzte Argument nimmt die Adresse einer Callback-Funktion auf, die von MCI bei festzulegenden Statusänderungen aufgerufen wird. Dieses interessante Feature (z. B. für eine Fortschrittsmeldung) habe ich aus Platzgründen aber nicht implementiert. `Open` übergibt hier als `IntPtr.Zero`.

Tritt bei der Ausführung von `mciSendString` ein Fehler auf, wird dieser über `GetMciError` in eine Fehlermeldung umgesetzt und an eine neue `MciException` übergeben.

Positions- und Längenangaben in MCI-Geräten verwenden ein bestimmtes Zeitformat. Bei den meisten Geräten sind das Millisekunden. Es kann aber bei spezifischen Geräten vorkommen, dass das Zeitformat ein anderes ist, wie z. B. eine Angabe in Stunden, Minuten und Sekunden. Um sicherzustellen, dass möglichst immer Millisekunden verwendet werden, stellt `Open` schließlich noch das aktuelle Zeitformat über den `set`-Befehl ein.

```
public void Open(string filename)
{
   Open(filename, null);
}

public void Open(string filename, Control owner)
{
   // Schließen einer eventuell zuvor noch geöffneten
   // Datei
   if (this.IsOpen)
   {
      this.Close();
   }

   // Alias für das MCI-Gerät erzeugen
   this.alias = Guid.NewGuid().ToString("N");

   // Überprüfen, ob die Datei existiert
   if (File.Exists(filename) == false)
   {
      throw new FileNotFoundException("Die Datei '" + filename +
         "' existiert nicht", filename);
   }
```

Listing 603: Methoden zum Öffnen eines MCI-Geräts

```
      // Den kurzen Dateinamen ermitteln
      StringBuilder shortPath = new StringBuilder(261);
      if (GetShortPathName(filename, shortPath, shortPath.Capacity) == 0)
      {
         throw new MciException("Fehler beim Auslesen des kurzen " +
            "Dateinamens für '" + filename + "': Windows-Fehler " +
            Marshal.GetLastWin32Error());
      }

      // MCI-Befehlsstring zum Öffnen zusammensetzen
      string mciString = "open " + shortPath.ToString() +
         " type mpegvideo alias " + this.alias;

      if (owner != null)
      {
         mciString += " parent " + (int)owner.Handle + " style child";
      }

      // MCI-Gerät öffnen
      int result = mciSendString(mciString, null, 0, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Öffnen des MCI-Geräts: " +
            GetMciError(result));
      }

      this.isOpen = true;

      // Das Zeitformat für Längen- und Positionsangaben explizit
      // auf Millisekunden setzen
      mciString = "set " + this.alias + " time format ms";
      result = mciSendString(mciString, null, 0, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Setzen des Zeitformats: " +
            GetMciError(result));
      }
   }
```

Listing 603: Methoden zum Öffnen eines MCI-Geräts (Forts.)

Nun folgt zunächst eine Eigenschaft `Length`, die über den MCI-Befehlsstring "`status` *Alias* `length`" die Abspiellänge der Datei (im aktuellen Zeitformat) ermittelt. Dazu wird natürlich wieder `mciSendString` aufgerufen, dieses Mal aber mit einem ausreichend groß dimensionierten `StringBuilder`-Objekt, in das `mciSendString` das Ergebnis schreibt. Tritt beim Aufruf kein Fehler auf, wird das Ergebnis über `int.Parse` in einen `int`-Wert umgewandelt. Ich verwende explizit die `Parse`-Methode des `int`-Typs, da die Länge für spezifische Gerätetypen laut der

Dokumentation trotz der expliziten Definition in `Open` bei einigen wenigen Medientypen auch in einer nicht numerischen Form zurückgegeben werden kann. Eine Konvertierung mit `Convert.ToInt32` würde in diesem Fall fehlschlagen.

```
public int Length
{
   get
   {
      StringBuilder buffer = new StringBuilder(261);
      int result = mciSendString("status " + this.alias + " length",
         buffer, buffer.Capacity, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Lesen von 'Length': " +
            GetMciError(result));
      }

      return int.Parse(buffer.ToString());
   }
}
```

Listing 604: Eigenschaft zur Ermittlung der Abspiellänge

Die nun folgende `Play`-Methode zum Abspielen der geöffneten Datei ist in drei Varianten implementiert, wobei die erste und die zweite jeweils die dritte aufrufen. Diese erwartet eine Startposition (in der aktuellen Längeneinheit, die wir ja auf Millisekunden gesetzt haben), eine Länge und eine Info darüber, ob der Abspielvorgang am Ende wiederholt werden soll. Der eigentliche Aufruf des MCI-Befehls entspricht dann dem beim Öffnen, nur dass hier natürlich ein anderer Befehlsstring verwendet wird. Die ersten beiden Varianten der Methode fragen über die `Length`-Eigenschaft einfach die aktuelle Länge der Datei ab um die Abspiellänge zu ermitteln.


```
public void Play(bool repeat)
{
   Play(0, this.Length, repeat);
}

/* Spielt eine geladene Multimediadatei ab einer bestimmten Position ab */
public void Play(int from, bool repeat)
{
   Play(from, this.Length - from, repeat);
}
```

Listing 605: Methoden zum Abspielen der Multimedia-Datei

```
/* Spielt eine geladene Multimediadatei ab einer bestimmten Position
   und bis zu einer bestimmten anderen Position ab */
public void Play(int from, int to, bool repeat)
{
   string mciString = "play " + this.alias +
      " from " + from + " to " + to;
   if (repeat)
   {
      mciString += " repeat";
   }
   int result = mciSendString(mciString, null, 0, IntPtr.Zero);
   if (result != 0)
   {
      throw new MciException("Fehler beim Aufruf von 'Play': " +
         GetMciError(result));
   }
}
```

Listing 605: Methoden zum Abspielen der Multimedia-Datei (Forts.)

Die Eigenschaft `Volume` ermöglicht das Abfragen und Setzen der aktuellen Lautstärke. Die Abfrage erfolgt über den MCI-Befehl `status`, das Setzen über den Befehl `setaudio`. Die in meinen Versuchen möglichen Werte reichten von 0 bis 1.000, leider ist dies aber nicht dokumentiert. Bei einem Wert von 1.000 entspricht die Lautstärke der, die aktuell für den Soundtreiber eingestellt ist. Bei einem ungültigen Wert generiert `mciSendString` einen Fehler, der wie in allen anderen Methoden dieser Klasse auch in eine Ausnahme umgesetzt wird.

```
public int Volume
{
   get
   {
      StringBuilder buffer = new StringBuilder(261);
      int result = mciSendString("status " + this.alias + " volume",
         buffer, buffer.Capacity, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Lesen von 'Volume': " +
            GetMciError(result));
      }
      return int.Parse(buffer.ToString());
   }

   set
   {
      int result = mciSendString("setaudio " + this.alias +
```

Listing 606: Eigenschaft für die Lautstärke

```
            " volume to " + value, null, 0, IntPtr.Zero);
         if (result != 0)
         {
            throw new MciException("Fehler beim Aufruf von 'SetAudio': " +
               GetMciError(result));
         }
      }
   }
```

Listing 606: Eigenschaft für die Lautstärke (Forts.)

Über die Eigenschaft `Position` können Sie die aktuelle Position im Abspiel-Stream lesen und setzen. Die Arbeitsweise dieser Eigenschaft ist ähnlich der von `Volume`. Nach dem Setzen der neuen Position ruft Position den `Play`-Befehl auf, da der `Seek`-Befehl dazu führt, dass der Abspielvorgang angehalten wird. So können Sie auch mitten im Abspielen vor- und zurückspulen. Als Nebeneffekt bewirkt dies, dass der Abspielvorgang auch dann beginnt, wenn aktuell noch keine `Play`-Methode aufgerufen wurde, aber die Position gesetzt wird.

```
public int Position
{
   get
   {
      StringBuilder buffer = new StringBuilder(261);
      int result = mciSendString("status " + this.alias + " position",
         buffer, buffer.Capacity, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Lesen von 'Position': " +
            GetMciError(result));
      }
      return int.Parse(buffer.ToString());
   }

   set
   {
      int result = mciSendString("seek " + this.alias +
         " to " + value, null, 0, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Setzen von 'Position'" +
            GetMciError(result));
      }
      result = mciSendString("play " + this.alias, null, 0, IntPtr.Zero);
      if (result != 0)
      {
```

Listing 607: Eigenschaft für die Position des aktuellen Abspielvorgangs

```
            throw new MciException("Fehler beim Aufruf von 'Play': " +
               GetMciError(result));
         }
      }
   }
```

Listing 607: Eigenschaft für die Position des aktuellen Abspielvorgangs (Forts.)

Die Eigenschaft `PlaybackSpeed` verwaltet die aktuelle Abspielgeschwindigkeit. Der Wert 1.000 steht für die normale Geschwindigkeit. Ein Setzen der Geschwindigkeit führte in meinen Versuchen mit MP3- und MPEG-Dateien allerdings nicht zu einem Ergebnis.

```
public int PlaybackSpeed
{
   get
   {
      StringBuilder buffer = new StringBuilder(261);
      int result = mciSendString("status " + this.alias + " speed",
         buffer, buffer.Capacity, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Lesen von 'Speed'" +
            GetMciError(result));
      }
      return int.Parse(buffer.ToString());
   }

   set
   {
      string mciString = "set " + this.alias + " speed " + value;
      int result = mciSendString(mciString, null, 0, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Setzen von 'Speed': " +
            GetMciError(result));
      }
   }
}
```

Listing 608: Eigenschaft zur Verwaltung der Abspielgeschwindigkeit

Nun folgt die Methode `SetRectangle`, über die Sie die Position eines auf einem Formular oder Steuerelement abgespielten Videos bestimmen können. Diese Methode ist z. B. dann interessant, wenn Sie die Größe des Videos bei der Größenänderung eines Formulars anpassen wollen. Übergeben werden die x- und y-Position, die Breite und die Höhe. `SetRectangle` ruft zur Definition der Position und der Größe den MCI-Befehl "`put` *Alias* `window at` *x y Breite Höhe*" auf:

```
public void SetRectangle(int x, int y, int width, int height)
{
   string mciString = "put " + this.alias + " window at " + x +
      " " + y + " " + width + " " + height;
   int result = mciSendString(mciString, null, 0, IntPtr.Zero);
   if (result != 0)
   {
      throw new MciException("Fehler beim Aufruf von 'Put Window at': " +
         GetMciError(result));
   }
}
```

Listing 609: Methode zur Definition der Position und der Größe des Videos bei einer Ausgabe auf einem Formular oder Steuerelement

Schließlich folgen nur noch eine Methode `Stop` zum Stoppen des aktuellen Abspielvorgangs, eine Methode `Close` zum Schließen des MCI-Geräts, die `Dispose`-Methode (der `IDisposable`-Schnittstelle) und der Destruktor. `Stop` und `Close` rufen die entsprechenden MCI-Befehle auf, wobei `Close` noch das Feld `isOpen` zurücksetzt. `Dispose` und der Finalisierer rufen einfach nur `Close` auf:

```
/* Stoppt den aktuellen Abspielvorgang */
public void Stop()
{
   string mciString = "stop " + this.alias;
   int result = mciSendString(mciString, null, 0, IntPtr.Zero);
   if (result != 0)
   {
      throw new MciException("Fehler beim Aufruf von 'Stop': " +
         GetMciError(result));
   }
}

/* Schließt ein aktuell geöffnetes MCI-Gerät */
public void Close()
{
   if (this.isOpen)
   {
      string mciString = "close " + this.alias;
      int result = mciSendString(mciString, null, 0, IntPtr.Zero);
      if (result != 0)
      {
         throw new MciException("Fehler beim Aufruf von 'Close': " +
            GetMciError(result));
```

Listing 610: Methoden zum Stoppen, zum Schließen, die Dispose-Methode und der Destruktor

```
            }

            this.isOpen = false;
         }
      }

      /* Schließt ein aktuell geöffnetes MCI-Gerät */
      public void Dispose()
      {
         this.Close();
      }

      /* Finalisierer */
      ~Mci()
      {
         this.Close();
      }
}
```

Listing 610: Methoden zum Stoppen, zum Schließen, die Dispose-Methode und der Destruktor (Forts.)

Den Finalisierer und die `Dispose`-Methode habe ich zur Sicherheit implementiert, damit beim Ungültigwerden einer `Mci`-Instanz auf jeden Fall alle MCI-Ressourcen freigegeben werden. `Dispose` wird, wie Sie ja sicherlich wissen, automatisch aufgerufen, wenn die Instanz innerhalb einer `using`-Anweisung erzeugt wird:

```
using (Mci mci = new Mci())
{
   mci.Open("C:\\Videos\\axe.mpeg");
   mci.Play(false);
   MessageBox.Show("Playing ...");
}
```

Wenn Sie Objekte in einer `using`-Anweisung instanzieren, ist sichergestellt, dass diese beim Ungültigwerden immer die `Dispose`-Methode aufrufen und die Ressourcen unabhängig vom Garbage Collector freigeben.

Ein kleines Beispielprogramm (Abbildung 1721) soll die Anwendung der `Mci`-Klasse erläutern. Das Programm soll Audio- und Videodateien abspielen.

Das Formular erzeugt im `Load`-Ereignis eine Instanz der `Mci`-Klasse, was ich hier nicht darstelle. Die Referenz wird in dem privaten Feld `mci` verwaltet. In den im Folgenden beschriebenen Methoden verzichte ich im Buch aus Platz- und Übersichtsgründen auch auf die Ausnahmebehandlung (die Sie im Beispiel auf der CD aber vorfinden).

In der Ereignismethode für das `Click`-Ereignis des ÖFFNEN-Schalters wird das MCI-Gerät geöffnet. Ist die obere `CheckBox` eingeschaltet, wird als Handle für die Ausgabe eines Videos das Handle eines `PictureBox`-Steuerelements übergeben, das auf dem Formular angelegt ist. In diesem Fall erfolgt eine Videoausgabe auf dem Steuerelement. Wird kein Handle übergeben,

erzeugt MCI für die Ausgabe ein eigenes Fenster. Sounddateien werden natürlich nicht in einem Fenster ausgegeben. Die Übergabe eines Handles führt aber hier nicht zu einem Fehler.



Abbildung 172: Abspielen eines AVI-Videos in einem PictureBox-Steuerelement


```
private void btnOpen_Click(object sender, System.EventArgs e)
{
   try
   {
      // MCI-Gerät öffnen
      if (this.openVideoInFormCheckBox.Checked)
      {
         this.mci.Open(filenameTextBox.Text, this.pictureBox);
      }
      else
      {
         this.mci.Open(filenameTextBox.Text);
      }
```

Listing 611: Öffnen des MCI-Geräts mit der angegebenen Datei

```
      // Länge, aktuelle Lautstärke und aktuelle
      // Abspielgeschwindigkeit auslesen
      infoLabel.Text = "Länge: " + this.mci.Length + " ms";
      this.volumeTextBox.Text = this.mci.Volume.ToString();
      this.playbackSpeedTextBox.Text = this.mci.PlaybackSpeed.ToString();
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 611: Öffnen des MCI-Geräts mit der angegebenen Datei (Forts.)

Die Methode für den ABSPIELEN-Schalter definiert zunächst das Rechteck, auf dem die Multimedia-Datei ausgegeben werden soll, wenn die entsprechende `CheckBox` eingeschaltet ist. Dann ruft `btnPlay_Click` die `Play`-Methode auf, wobei der Zustand der zweiten `CheckBox` am `repeat`-Argument übergeben wird:

```
private void btnPlay_Click(object sender, System.EventArgs e)
{
   try
   {
      // Rechteck setzen, falls die Datei in der
      // PictureBox angezeigt werden soll
      if (this.openVideoInFormCheckBox.Checked)
      {
         this.mci.SetRectangle(0, 0,
             this.pictureBox.Width, this.pictureBox.Height);
      }

      // MCI-Gerät öffnen
      this.mci.Play(this.repeatCheckBox.Checked);
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 612: Abspielen der geöffneten Multimedia-Datei

Die Methoden für den LAUTSTÄRKE- und den VORSPULEN-Schalter lesen bzw. schreiben die Eigenschaften `Volume` und `Position`:

```
private void btnVolume_Click(object sender, System.EventArgs e)
{
   try
   {
      this.mci.Volume = Convert.ToInt32(this.volumeTextBox.Text);
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}

private void forwardButton_Click(object sender, System.EventArgs e)
{
   try
   {
      int position = this.mci.Position;
      this.infoLabel.Text = "Aktuelle Position: " + position;
      this.mci.Position = position + 10000;
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 613: Einstellen der Lautstärke und der Position

Die Methoden des STOP- und des SCHLIEßEN-Schalters rufen die Methoden `Stop` und `Close` auf:

```
private void btnStop_Click(object sender, System.EventArgs e)
{
   try
   {
      this.mci.Stop();
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 614: Stoppen und Schließen des aktuellen Abspielvorgangs

```
private void closeButton_Click(object sender, System.EventArgs e)
{
   try
   {
      this.mci.Close();
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}
```

Listing 614: Stoppen und Schließen des aktuellen Abspielvorgangs (Forts.)

Die Methode für das `Resize`-Ereignis des Formulars ruft `SetRectangle` auf, um das Video an die Größe der `PictureBox` anzupassen (die im Beispiel über die `Anchor`-Eigenschaft automatisch an die Größe des Formulars angepasst wird). Dabei fragt die Methode zuvor ab, ob das MCI-Gerät überhaupt geöffnet ist, um einen Fehler beim Aufruf des MCI-Befehls bei einem nicht geöffneten Gerät zu vermeiden (das könnten die anderen Methoden dieses Beispiels natürlich auch so implementieren):

```
private void StartForm_Resize(object sender, System.EventArgs e)
{
   if (this.mci.IsOpen)
   {
      try
      {
         this.mci.SetRectangle(0, 0, this.pictureBox.Width,
            this.pictureBox.Height);
      }
      catch { }
   }
}
```

Listing 615: Anpassen der Videogröße im Resize-Ereignis

Im `Closing`-Ereignis des Formulars wird die `Close`-Methode noch einmal aufgerufen:

```
private void StartForm_Closing(object sender,
   System.ComponentModel.CancelEventArgs e)
{
   this.mci.Close();
}
```

Listing 616: Schließen des MCI-Geräts im Closing-Ereignis

Dieses Schließen ist deswegen wichtig, weil MCI das MCI-Gerät scheinbar automatisch schließt, wenn der aufrufende Thread beendet wird. Wird die `Close`-Methode zuvor nicht aufgerufen, führt der spätere Aufruf des Destruktors über den Garbage Collector zu dem Fehler »Gerät ist nicht geöffnet oder wird vom MCI nicht erkannt«. Dieses Problem konnte ich bisher nicht lösen.

Die `Mci`-Klasse kann durchaus noch erweitert werden. Die folgenden Erweiterungen wären sinnvoll (und machbar):

- Regelmäßige Benachrichtigung über die aktuelle Position in einem Delegaten,
- separate Einstellung der Lautstärke für den linken und den rechten Kanal,
- Ein- und Ausschalten der Soundwiedergabe,
- Pausieren und Weiter-Ausführen der Wiedergabe,
- Einstellen der Bass- und der Höhenwiedergabe,
- Wiedergabe von CD-Audio-Tracks.

Drei bekannte Probleme sind in der Buch-Variante nicht gelöst:

- Wird eine neue Datei geöffnet, während eine andere abgespielt wird, werden die Dateien zwar parallel abgespielt (was ein interessantes Feature ist, so können Sie z. B. neben einem Video eine MP3-Datei oder zwei MP3-Dateien gleichzeitig abspielen), da aber nur ein Alias verwaltet wird, kann nur die letzte bearbeitet (z. B. geschlossen) werden.
- MCI funktionierte bei meinen Versuchen nicht in einer Konsolenanwendung. Obwohl kein Fehler erzeugt wurde, spielte MCI Multimedia-Dateien einfach nicht ab.
- Ist keine Soundkarte installiert, resultiert dies in der wenig aussagekräftigen Ausnahme. Auf meinem Testsystem meldete MCI den Fehler »MMSYSTEM277 Bei der Initialisierung von MCI trat ein Fehler auf. Starten Sie Windows neu«. Sinnvoll wäre es, in der `Open`-Methode zu überprüfen, ob auf dem System eine Soundkarte installiert ist, und im negativen Fall eine deutlichere Ausnahme zu werfen. Diese Überprüfung können Sie über das Rezept 197 vornehmen.

315 Multimediadateien in einer WPF-Anwendung abspielen

In einer WPF-Anwendung steht das `MediaElement`-Steuerelement zum Abspielen verschiedener Multimediadateien zur Verfügung. Die Verwendung dieses Steuerelements ist sehr einfach.

Zum Abspielen einer Multimediadatei weisen Sie der Eigenschaft `Source` einen `Uri` mit dem Pfad zur Datei zu. Abhängig von der Einstellung in der Eigenschaft `LoadedBehavior` wird die Datei sofort abgespielt (`Play`), pausiert (`Pause`) etc. Die Einstellung `Manual` führt dazu, dass Sie die `Play`-Methode explizit aufrufen müssen, um die Multimediadatei abzuspielen.

Über die Methoden `Pause` und `Stop` können Sie das Abspielen pausieren bzw. beenden. Die Eigenschaft `UnloadedBehavior` gibt an, was passiert, wenn die Multimediadatei entladen wird. Hier geben Sie üblicherweise `Stop` an, damit das Abspielen beendet wird.

Für das Abspielen von Videos spielt die Eigenschaft `Stretch` eine Rolle, die Sie auf die Werte `None` (keine Anpassung der Videogröße), `Fill` (Anpassung der Videogröße an das `MediaElement`), `Uniform` (Anpassung der Videogröße an das `MediaElement` unter Beibehaltung der Proportionen) und `UniformToFill` (Anpassung der Videogröße an das `MediaElement` unter Beibehaltung der Proportionen mit Auffüllen des `MediaElement`-Steuerelements und Abschneiden überstehender Ränder) setzen können.

Daneben können Sie u. a. noch über die Eigenschaft `Volume` die Lautstärke einstellen (im Bereich von 0 bis 1, Default ist 0,5) und über die Ereignisse `MediaOpened` und `MediaEnded` das Öffnen und das Beenden des Abspielens ermitteln.

Das Beispiel in Listing 617 zeigt eine typische Anwendung. Es basiert auf einem Fenster, das ein `MediaElement`, vier Schalter (*openButton*, *playButton*, *pauseButton* und *stopButton*) und einen `Slider` (*volumeSlider*, zur Einstellung der Lautstärke) enthält.

Das `MediaElement` ist folgendermaßen deklariert:

```
<MediaElement Margin="100,50,10,20" Name="mediaElement"
   LoadedBehavior="Manual" UnloadedBehavior="Stop" Stretch="Uniform" />
```

Das Beispiel erfordert die für WPF üblichen `using`-Deklarationen.

```
/* Öffnet die angegebene Mediadatei */
private void openButton_Click(object sender, RoutedEventArgs e)
{
   string fileName = "C:\\Demo.mp3";
   this.mediaElement.Source = new Uri(fileName);
}

/* Spielt die aktuelle Mediadatei ab */
private void playButton_Click(object sender, RoutedEventArgs e)
{
   this.mediaElement.Play();
}

/* Pausiert das aktuelle Abspielen */
private void pauseButton_Click(object sender, RoutedEventArgs e)
{
   this.mediaElement.Pause();
}

/* Beendet das aktuelle Abspielen */
private void stopButton_Click(object sender, RoutedEventArgs e)
{
   this.mediaElement.Stop();
}
```

Listing 617: Abspielen einer Multimediadatei in einer WPF-Anwendung

```
/* Stellt die Lautstärke ein */
private void volumeSlider_ValueChanged(object sender,
   RoutedPropertyChangedEventArgs<double> e)
{
   this.mediaElement.Volume = this.volumeSlider.Value;
}
```

Listing 617: Abspielen einer Multimediadatei in einer WPF-Anwendung (Forts.)

Bildbearbeitung

316 Speicherschonend in einer Windows.Forms-Anwendung mit Bildern umgehen

Bei der Arbeit mit `System.Drawing.Bitmap`-Objekten, die Sie vorwiegend in einer `Windows.Forms`-Anwendung einsetzen, reservieren die CLR und Windows relativ viel Speicher für das ausführende Programm. `Bitmap`-Objekte werden normalerweise vom Garbage Collector relativ schnell wieder freigegeben. Der Finalisierer der Basisklasse `Image` ruft dabei die `Dispose`-Methode auf, die die externen reservierten Ressourcen freigibt. Wenn Sie aber mehrere Bitmaps in einer Schleife verarbeiten, z. B. weil Ihr Programm alle Bilder aus einem Ordner ausliest, diese verkleinert und in einem anderen Ordner abspeichert, erhält der Garbage Collector keine Zeit, seine Arbeit zu verrichten. Während des Ablaufs der Schleife steigt der reservierte Speicher des Programms teilweise enorm an.

Hinweis

In einer WPF-Anwendung, die nicht auf GDI+ aufsetzt, sondern eine eigene Bildverarbeitungs-Engine besitzt, tritt dieses Problem allerdings nicht auf.

Das folgende Beispiel liest in einer `Windows.Forms`-Anwendung eine Bilddatei in einer Schleife einhundert Mal ein und erzeugt ein einfaches Thumbnail-Bild:

```
for (int i = 0; i < 100; i++)
{
   // Bitmap laden
   string filename = Path.Combine(Application.StartupPath, "Gardasee.jpg");
   Bitmap bitmap = new Bitmap(filename);

   // Bitmap verarbeiten
   Image thumbnail = bitmap.GetThumbnailImage(100, 50, null, (IntPtr)0);

   // Thumbnail speichern (nicht implementiert)
}
```

Listing 618: Simulation der (nicht speicherschonenden) Bearbeitung mehrerer Bilder in einer Schleife

Auf meinem Rechner stieg der reservierte Speicher während der Ausführung der Schleife teilweise bis über 350 MB und wurde nur relativ selten wieder freigegeben. In einigen Tests blieb der reservierte Speicher sogar nach der Beendigung der Schleife auf einem relativ hohen Wert (größer als 200 MB) stehen. Wie ich in Rezept 85 genauer zeige, handelt es sich bei dem im Task-Manager angezeigten Speicher nicht um den tatsächlich verbrauchten, sondern lediglich um den von der CLR angeforderten und von Windows reservierten Speicher. Dieses Rezept zeigt auch, wie Sie den reservierten Speicher reduzieren können.

Normalerweise sollten auch bei solch hohen Speicherwerten keine Probleme auftreten, da die CLR gemeinsam mit Windows für eine optimale Speicherverteilung an die einzelnen Anwendungen sorgt. Trotzdem kann es in einigen wenigen Fällen vorkommen, dass zumindest das gesamte System verlangsamt wird, weil Windows zu viel Speicher auslagern muss.

Deshalb sollten Sie so programmieren, dass Sie die Ressourcen möglichst schnell wieder freigeben. Und dazu können Sie einfach die `Dispose`-Methode der verwendeten `Bitmap`-Objekte aufrufen. Idealerweise verwenden Sie dazu eine `using`-Anweisung, die `Dispose` automatisch am Ende aufruft:

```
// Bitmap laden
string filename = Path.Combine(Application.StartupPath, "Gardasee.jpg");
using (Bitmap bitmap = new Bitmap(filename))
{
   // Bitmap verarbeiten
   using (Image thumbnail = bitmap.GetThumbnailImage(
      100, 50, null, (IntPtr)0))
   {
      // Thumbnail speichern (nicht implementiert)
   }
}
```

Listing 619: Speicherschonendes Lesen und Verarbeiten von Bildern

Das Resultat dieser kleinen Änderung war in meinem Fall, dass das Programm nur noch maximal 16 MB Speicher reservierte.

Die `using`-Anweisung hat übrigens den Effekt, dass der Compiler im erzeugten CIL-Code einen sicheren Aufruf von `Dispose` einbaut:

```
using (Bitmap bitmap = new Bitmap(filename))
{
   // Bitmap verarbeiten
}
```

entspricht:

```
Bitmap bitmap = null;
try
{
   bitmap = new Bitmap(filename);

   // Bitmap verarbeiten
}
finally
{
   if (bitmap != null)
   {
      bitmap.Dispose();
   }
}
```

Wie gesagt tritt das Ressourcen-Freigabe-Problem in WPF-Anwendungen nicht auf. Die Klasse `BitmapSource` besitzt deswegen auch gar keine `Dispose`-Methode. In einer WPF-Bildverarbeitungs-Demoanwendung, die Sie im Ordner des Beispiels zu diesem Rezept finden, können Sie dies nachvollziehen. Während der Ausführung eines Programmcodes, der dem in Listing 618 entspricht, wird der reservierte Speicher nicht erhöht.

317 Bitmap-Objekte aus BitmapSource-Objekten erzeugen

Für den Fall, dass Sie in einer WPF-Anwendung mit `BitmapSource`-Objekten arbeiten, die Bilder, die diese beinhalten, aber in Form eines `Bitmap`-Objekts an Methoden oder Typen weitergeben müssen, hilft dieses Rezept. Das Erzeugen eines `Bitmap`-Objekts aus einem `BitmapSource`-Objekt ist nicht allzu einfach. Microsoft hat scheinbar keinen direkten Support dafür in das .NET Framework eingebaut. Robert A. Wlodarczyk, ein Microsoft-Mitarbeiter, der u. a. an WPF Imaging arbeitet, beschreibt in seinem Blog (*blogs.msdn.com/rwlodarc*), eine Lösung, die mit dem Kopieren der Pixeldaten in ein `Byte`-Array arbeitet (*blogs.msdn.com/rwlodarc/archive/2007/01/03/wpf-bitmapsource-and-gdi-bitmap-interop.aspx*). Das `Byte`-Array wird dann über einen Zeiger (in einem unsicheren Block) referenziert und zum Erzeugen eines neuen `Bitmap`-Objekts verwendet.

Roberts Lösung hat allerdings drei Probleme (sorry, Robert ...):

- Sie berücksichtigt nicht, dass das Pixelformat des `BitmapSource`-Objekts zum in der Lösung für das `Bitmap`-Objekt verwendeten Pixel-Format inkompatibel sein kann. Bei inkompatiblen Pixelformaten resultieren zerstörte Bilder.
- Sie verändert das Bild in der Größe um dies an die Tatsache anzupassen, dass WPF bei der Anzeige (!) von Bildern deren DPI-Wert berücksichtigt (also Bilder mit einem höheren DPI-Wert als der Monitor kleiner und Bilder mit einem kleineren DPI-Wert größer anzeigt). Ich denke allerdings, dass nicht das Bild angepasst werden sollte, da dieses wahrscheinlich eher nicht angezeigt, sondern an Methoden oder Eigenschaften übergeben werden soll. Außerdem halte ich von der Bildskalierung von WPF nach dem DPI-Wert nicht viel.
- Schließlich erfordert Roberts Lösung das Kompilieren mit der Option UNSICHEREN CODE ZULASSEN.

Ich habe deswegen eine (wahrscheinlich langsamere) Methode entwickelt, die auf den Rezepten 323 und 324 basiert. Diese Lösung erzeugt zunächst aus dem `BitmapSource`-Objekt einen `MemoryStream` im PNG-Format (in der Hoffnung, dass PNG alle Bilder 1:1 darstellen kann) und verwendet diesen dann zur Erzeugung eines `Bitmap`-Objekts.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System.Drawing`, `System.IO` und `System.Windows.Media.Imaging` importieren.


```
public static Bitmap BitmapSource2Bitmap(BitmapSource sourceImage)
{
   // Einen PNG-Encoder erzeugen
   PngBitmapEncoder encoder = new PngBitmapEncoder();
```

Listing 620: Methode zum Erzeugen eines Bitmap-Objekts aus einem BitmapSource-Objekt

```
    // Das Bild dem Encoder hinzufügen
    encoder.Frames.Add(BitmapFrame.Create(sourceImage));

    // MemoryStream erzeugen
    using (MemoryStream imageStream = new MemoryStream())
    {
        // Das Bild in den MemoryStream schreiben
        encoder.Save(imageStream);
        imageStream.Flush();
        // Mit dem MemoryStream ein Bitmap erzeugen
        imageStream.Position = 0;
        return new Bitmap(imageStream);
    }
}
```

Listing 620: Methode zum Erzeugen eines Bitmap-Objekts aus einem BitmapSource-Objekt (Forts.)

Hinweis

Das Beispiel zu diesem Rezept beweist, dass die Methode für die gängigen Bildformate und für unterschiedliche Paletten (z. B. auch Graustufen-Bilder) funktioniert. Mit dem Beispiel können Sie das Umwandeln auch für Ihre eigenen Bilder testen.

318 BitmapSource-Objekte aus Bitmap-Objekten erzeugen

Das Erzeugen eines `BitmapSource`-Objekts aus einem `Bitmap`-Objekt ist, verglichen mit dem umgekehrten Weg (siehe Rezept 317), relativ einfach. Dazu können Sie nämlich die (sehr spärlich dokumentierte) statische Methode `CreateBitmapSourceFromHBitmap` der `Imaging`-Klasse aus dem Namensraum `System.Windows.Interop` verwenden.

Am ersten Argument übergeben Sie einen Windows-Handle auf das GDI-Bild, den Sie über die `GetHbitmap`-Methode des `Bitmap`-Objekts auslesen können. Das zweite Argument ist ein Windows-Handle auf die Palette, die das Bild verwendet. Leider können Sie (über die Eigenschaft `Palette`) zwar die Palette auslesen, aber nicht den entsprechenden Windows-Handle. Also bleibt nichts anderes übrig, als an diesem Argument `IntPtr.Zero` zu übergeben (was auf magische Weise scheinbar funktioniert).

Am dritten Argument erwartet die Methode ein Rechteck, das einen zu kopierenden Ausschnitt kennzeichnet. Übergeben Sie hier `Int32Rect.Empty`, wird das gesamte Bild kopiert.

Das letzte Argument schließlich ist laut der Dokumentation für die Steuerung von Konvertierungen vorgesehen. Hier können Sie ein Objekt übergeben, das eine der statischen Methoden der `BitmapSizeOptions`-Klasse zurückgibt. Scheinbar können Sie damit z. B. erreichen, dass das Zielbild gleich skaliert (`FromWidth`-, `FromHeight`- und `FromWidthAndHeight`-Methode) oder rotiert (`FromRotation`-Methode) wird. Übergeben Sie das Objekt, das die `FromEmptyOptions`-Methode zurückgibt, erfolgt keine Konvertierung.

Die Methode `Bitmap2BitmapSource` in Listing 621 setzt dieses Wissen um. Sie erfordert den Import der Namensräume `System` und `System.Windows`.

```
public static System.Windows.Media.Imaging.BitmapSource
   Bitmap2BitmapSource(System.Drawing.Bitmap sourceImage)
{
   return System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap(
      sourceImage.GetHbitmap(), IntPtr.Zero, Int32Rect.Empty,
      System.Windows.Media.Imaging.BitmapSizeOptions.FromEmptyOptions());
}
```

Listing 621: Methode zum Erzeugen eines BitmapSource-Objekts aus einem Bitmap-Objekt

Hinweis

In dem Beispiel zu diesem Rezept können Sie die Umwandlung für die gängigen Bildformate und für unterschiedliche Paletten (z. B. auch Graustufen-Bilder) testen.

319 Das Format eines Bilds auslesen

Das Format eines Bilds spielt in einem Programm, in dem Sie das Bild über ein `BitmapSource`-(WPF) oder `Bitmap`-Objekt (`Windows.Forms`) verwalten, eigentlich keine Rolle. Diese Objekte verwalten ihre Daten immer Format-unabhängig. Trotzdem können Sie das Format (mehr oder weniger) im Programm auslesen.

WPF

In einer WPF-Anwendung können Sie das Format eines Bilds aus einem `BitmapDecoder`-Objekt auslesen, das Sie zum Einlesen des Bilds verwenden. Die Eigenschaft `CodecInfo` verwaltet Informationen zu `Codec`. Diese (unzureichend dokumentierte) Eigenschaft ist vom Typ `BitmapCodecInfo`. Die einzelnen Codecs werden leider nicht durch eigene Codec-Info-Klassen repräsentiert, obwohl `BitmapCodecInfo` abstrakt ist. Das liegt wahrscheinlich daran, dass auf einem System andere Codecs vorliegen können als auf einem anderen.

In der Eigenschaft `MimeTypes` dieses Objekts finden Sie eine Angabe der MIME-Typen, die dem Codec zugeordnet sind. Die möglichen (Standard-)MIME-Typen sind leider nicht dokumentiert. Ich habe die folgenden `MimeTypes`-Werte für die Standard-Bildformate herausgefunden:

Bildformat	Mime-Typ-Angabe
Bitmap	image/bmp
GIF	image/gif
JPEG	image/jpeg,image/jpe,image/jpg
TIFF	image/tiff,image/tif
PNG	image/png

Listing 622: Die von MimeTypes zurückgegebenen MIME-Typen für Standard-Bildformate

Über die Abfrage der MIME-Typen können Sie das Format also recht einfach auslesen. Da auch mehrere Angaben enthalten sind, fragt die Methode `GetImageFormatName` in Listing 623 einfach, ob die MIME-Typen (siehe *de.selfhtml.org/diverses/mimetypen.htm*) der am häufigsten vorkommenden Bildformate enthalten sind.

Zum Kompilieren dieser Methode müssen Sie den Namensraum `System.Windows.Media.Imaging` einbinden.

```
public static string GetImageFormatName(BitmapDecoder decoder)
{
   // Das Format des Bilds ermitteln
   if (decoder.CodecInfo.MimeTypes.Contains("image/bmp"))
   {
      return "Bitmap";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/gif"))
   {
      return "GIF";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/jpeg"))
   {
      return "JPEG";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/tiff"))
   {
      return "TIFF";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/png"))
   {
      return "PNG";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/fif"))
   {
      return "FIF";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/ief"))
   {
      return "IEF";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/vasa"))
   {
      return "Vasa";
   }
   else if (decoder.CodecInfo.MimeTypes.Contains("image/vnd.wap.wbmp"))
   {
      return "Bitmap (WAP)";
```

Listing 623: Methode zum Ermitteln des Bildformats in einer WPF-Anwendung

```
    }
    else if (decoder.CodecInfo.MimeTypes.Contains("image/x-icon"))
    {
        return "Icon";
    }
    else if (decoder.CodecInfo.MimeTypes.Contains("image/x-rgb"))
    {
        return "RGB";
    }
    else if (decoder.CodecInfo.MimeTypes.Contains("image/x-xbitmap"))
    {
        return "X-Bitmap";
    }
    else
    {
        return "Unbekannt";
    }
}
```

Listing 623: Methode zum Ermitteln des Bildformats in einer WPF-Anwendung (Forts.)

Listing 624 zeigt eine beispielhafte Anwendung dieser Methode.

```
string fileName = "C:\\Bilder\\Hitchhiker.gif";

// Bild in einen BitmapDecoder einlesen
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(fileName),
    BitmapCreateOptions.None, BitmapCacheOption.None);

// Das Format des Bilds ermitteln
string formatName = GetImageFormatName(decoder);

// Dateiname und Format ausgeben
Console.WriteLine(fileName + ": " + formatName);
```

Listing 624: Anwendung der Methode zur Abfrage des Bildformats in einer WPF-Anwendung

Hinweis

Interessant (und eigentlich auch logisch) ist, dass das Format eines Bilds nicht von der Dateiendung abhängt, sondern im Header der Bilddatei verwaltet wird. Wenn Sie z. B. die Endung einer GIF-Datei in *.png* ändern, wird die Datei trotzdem als GIF-Datei eingelesen. `GetImageFormatName` liefert dann auch wie erwartet »GIF« zurück.

Windows.Forms

Ein in einer `Windows.Forms`-Anwendung verwendetes `Bitmap`-Objekt verwaltet das originale Format eines Bilds, das z. B. aus einer Datei eingelesen wurde, in der Eigenschaft `RawFormat`.

Diese Eigenschaft spielt auch eine Rolle, wenn das in einem `Bitmap`-Objekt verwaltete Bild in eine Datei gespeichert wird (siehe Rezept 322).

In einer `Windows.Forms`-Anwendung können Sie das Originalformat eines Bilds also über `RawFormat` ermitteln. Diese Eigenschaft verwaltet eine Instanz der Klasse `System.Drawing.Imaging.ImageFormat`. Leider besitzt diese Klasse keine Eigenschaft, die den Namen des Formats zurückgibt. Die `ToString`-Methode liefert lediglich die GUID des System-Encoders für das Format mit vorangestelltem »ImageFormat:« und in eckigen Klammern. Für das GIF-Format liefert `ToString` (auf meinem System) z. B. »[ImageFormat: b96b3cb0-0728-11d3-9d7b-0000f81ef32e]«. Besonders viel kann ich damit nicht anfangen.

Die `ImageFormat`-Klasse bietet einige statische Eigenschaften, die Instanzen für die verschiedenen Bildformate zurückgeben. `ImageFormat.Bmp` liefert z. B. eine `ImageFormat`-Instanz für das `Bitmap`-Format. Leider lassen sich `ImageFormat`-Instanzen nicht über den Vergleichsoperator direkt miteinander vergleichen. Aber die `Equals`-Methode funktioniert wie erwartet. Über einen `Equals`-Vergleich der `RawFormat`-Eigenschaft mit `ImageFormat.Bmp` können Sie z. B. herausfinden, ob das Bild im Bitmap-Format gespeichert ist.

Die Methode `GetImageFormatName` in Listing 625 macht genau das. Zum Kompilieren dieser Methode müssen Sie den Namensraum `System.Drawing.Imaging` einbinden.

```
public static string GetImageFormatName(Bitmap bitmap)
{
   // Das Format ermitteln
   if (bitmap.RawFormat.Equals(ImageFormat.Bmp))
   {
      return "Bitmap";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Emf))
   {
      return "EMF";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Exif))
   {
      return "EXIF";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Gif))
   {
      return "GIF";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Jpeg))
   {
      return "JPEG";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.MemoryBmp))
   {
```

Listing 625: Methode zur Abfrage des Bildformats in einer Windows.Forms-Anwendung

```
      return "Speicher-Bitmap";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Png))
   {
      return "PNG";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Tiff))
   {
      return "TIFF";
   }
   else if (bitmap.RawFormat.Equals(ImageFormat.Wmf))
   {
      return "WMF";
   }
   else
   {
      return "Unbekanntes Format";
   }
}
```

Listing 625: Methode zur Abfrage des Bildformats in einer Windows.Forms-Anwendung (Forts.)

Listing 626 zeigt eine beispielhafte Anwendung dieser Methode.

```
string fileName = "C:\\Bilder\\Hitchhiker.gif";

// Bitmap erzeugen
using (Bitmap bitmap = new Bitmap(fileName))
{
   // Das Format ermitteln
   string formatName = GetImageFormatName(bitmap);

   // Dateiname und Format ausgeben
   Console.WriteLine(fileName + ": " + formatName);
};
```

Listing 626: Anwendung der Methode zur Abfrage des Bildformats in einer WPF-Anwendung

Hinweis

Der `else`-Block wird zurzeit noch nicht angesprungen, da die `ImageFormat`-Aufzählung nur die in dieser Methode verwendeten Konstanten besitzt. Für den (sehr unwahrscheinlichen) Fall, dass GDI+ aber in Zukunft neue Bildformate (z. B. JPEG2000) unterstützt, ist dieser Block wichtig. In diesem Fall sollten Sie den Quellcode natürlich entsprechend erweitern.

320 Bild-Metadaten auslesen

Viele Grafikformate verwalten nicht nur das eigentliche Bild, sondern auch zusätzliche Informationen zum Bild. Das JPEG-Format ist beispielsweise u. a. in der Lage, das Aufnahmedatum eines mit einer Digitalkamera aufgenommenen Bilds zu speichern. Diese Zusatzinformationen, die als *Image Tag* (Bild-Etikett) bezeichnet werden, können Sie in WPF- und in `Windows.Forms`-Anwendungen auswerten.

Grundlagen

Metadaten werden von verschiedenen Programmen in Bilder geschrieben. Eine Digitalkamera speichert z. B. das Aufnahmedatum, das Kameramodell um den Hersteller in den erzeugten Bildern. Die Nachrichtenagentur Reuters speichert Copyright-Informationen in den online veröffentlichten Bildern. Windows Vista zeigt diese Metadaten übrigens im Explorer an (wobei dieser die WPF-Klassen zum auslesen der Metainformationen verwendet).

Metadaten können in verschiedenen Standards (auch mehrere pro Bild) in einem Bild gespeichert sein. WPF unterstützt die Standards *Exchangeable Image Format* (EXIF, z. B. in JPEG, TIFF- und JFIF-Dateien), *International Press Telecommunications Council* (IPTC, z. B. in TIFF- und JFIF-Dateien), *Extensible Metadata Platform* (XMP, in PDF-Dateien), *PNG Textual Data* (tEXt, in PNG-Dateien) und *Image File Directory* (IFD, in TIFF-Dateien).

Nähere Informationen zu den Standards finden Sie größtenteils bei Wikipedia:

- EXIF: *de.wikipedia.org/wiki/EXIF*
- IPTC: *de.wikipedia.org/wiki/IPTC-NAA-Standard*
- XMP: *de.wikipedia.org/wiki/Extensible_Metadata_Platform*
- tEXt: *www.w3.org/TR/PNG-Chunks.html*
- IFD: *de.wikipedia.org/wiki/Tagged_Image_File_Format*

In dem zu `Windows.Forms` gehörenden GDI+ wird scheinbar nur der EXIF-Standard unterstützt (was aber nicht dokumentiert ist). Auf diesen Standard gehe ich deswegen auch näher ein.

EXIF ist ein Standard der *Japan Electronic Industry Development Association* (JEIDA) und beschreibt ein spezielles Dateiformat für Digitalkameras, in dem neben dem eigentlichen Bild zusätzliche Metadaten gespeichert werden können. Als Bildformat wird bei EXIF JPEG oder TIFF verwendet. Informationen zu EXIF finden Sie an der Adresse *www.exif.org/specifications.html*.

Die einzelnen Bild-Tags können unterschiedliche Datentypen besitzen, die (natürlich) im EXIF-Standard beschrieben sind. Tabelle 34 beschreibt diese Typen.

Datentyp	Beschreibung
`BYTE`	Byte
`ASCII`	Nullterminierter 7-Bit-ASCII-String
`SHORT`	16-Bit Integer ohne Vorzeichen (!)
`LONG`	32-Bit (!) Integer ohne Vorzeichen (!)

Tabelle 34: Die Datentypen des EXIF-Standards

Datentyp	Beschreibung
RATIONAL	RATIONAL besteht aus zwei LONG-Werten, die zusammen eine rationale Zahl ergeben. Der erste verwaltet den Zähler, der zweite den Nenner. Das Ergebnis eines RATIONAL-Wertes ist also die Division des ersten durch den zweiten Wert.
UNDEFINED	Byte-Wert, der alle Werte annehmen kann, abhängig von der Tag-Beschreibung
SLONG	32-Bit (!) Integer mit Vorzeichen
SRATIONAL	SRATIONAL ist RATIONAL ähnlich, besteht aber aus zwei SLONG-Werten.

Tabelle 34: Die Datentypen des EXIF-Standards (Forts.)

Die Werte aller Datentypen werden in Arrays verwaltet. Pro Image Tag sind also prinzipiell mehrere Werte möglich. ASCII-Tags verwalten einzelne Zeichen (in einem Byte-Array), alle anderen ein Array aus dem jeweiligen Datentyp. Der EXIF-Standard beschreibt für jedes Tag, wie viele Werte verwaltet werden. Meist handelt es sich nur um einen Wert, in diesem Fall müssen Sie also nur das erste Arrayelement auswerten. Bei einigen wenigen Tags, wie z. B. ISOSpeedRatings (ISO-Geschwindigkeiten der Kamera) können auch mehrere Werte verwaltet werden.

Das Ganze ist also nicht allzu einfach auszulesen. WPF erleichtert die Abfrage (und das in diesem Rezept nicht behandelte Schreiben) von Metadaten aber erheblich.

WPF

In einer WPF-Anwendung (oder einer Anwendung, die die WPF-Klassen verwendet ...) stehen Metadaten über die Eigenschaft Metadata einer BitmapSource-Instanz zur Verfügung. Diese Eigenschaft verwaltet nur dann eine Instanz einer von ImageMetadata abgeleiteten Klasse, wenn das Bild Metadaten enthält. ImageMetadata selbst ist abstrakt und stellt nur Basis-Funktionalitäten zur Verfügung. Die zurzeit (.NET 3.5, erstes Release) einzige von ImageMetadata abgeleitete Klasse ist BitmapMetadata, und diese stellt einige Eigenschaften zur Verfügung, über die die Standard-Metadaten auslesen können (Tabelle 35).

Eigenschaft	Beschreibung
ApplicationName	Name der Anwendung, mit der das Bild ggf. erzeugt wurde
Author	ReadOnlyCollection<String> mit den Autoren des Bilds
Camera-Manufacturer	Der Hersteller der Kamera, mit der das Bild ggf. aufgenommen wurde
CameraModel	Das Kameramodell, mit dem das Bild ggf. aufgenommen wurde
Comment	String mit Kommentaren
Copyright	Eine Copyright-Meldung
DataTaken	Das Aufnahmedatum als String. Das Format ist leider nicht dokumentiert. Mit meinen Testbildern erhielt ich das Datum immer in dem Format der Kultur, die für den aktuellen Thread eingestellt war. Ob das immer so ist, kann ich leider nicht sagen, es muss ja schließlich einen Sinn haben, dass hier (dummerweise) ein String zurückgegeben wird.
Format	Das Format des Bilds als String, der die für das Format typische Dateiendung enthält

Tabelle 35: Die Metadaten-Eigenschaften der BitmapMetadata-Klasse

Eigenschaft	Beschreibung
Keywords	ReadOnlyCollection<String> mit Schlüsselwörtern, die dem Bild zugeordnet sind
Rating	Integer-Wert, der eine Bewertung des Bilds enthält. In Vista können Sie Bilder z. B. im Explorer mit einem Wert von 0 bis 5 bewerten.
Subject	Beschreibung des Themas des Bilds
Title	Der Titel des Bilds

Tabelle 35: Die Metadaten-Eigenschaften der BitmapMetadata-Klasse (Forts.)

Andere Metadaten, die nicht zum »Standard« gehören, können Sie über die GetQuery-Methode der BitmapMetadata-Klasse auslesen. Soweit will ich aber nicht gehen, und zeige nur, wie Sie die Standard-Daten auslesen. Das Beispiel erfordert den Import der Namensräume System und System.Windows.Media.Imaging.

```
// BitmapSource-Objekt für das Bild erzeugen
string fileName = "C:\\Bilder\\Windsurfen.jpg";
BitmapSource bitmapSource = BitmapFrame.Create(new Uri(fileName));

// Ermitteln, ob Metadaten vorhanden sind
if (bitmapSource.Metadata != null)
{
   // Ermitteln, ob die Metadaten vom Typ BitmapMetadata sind
   // (die einzige zurzeit von ImageMetadata abgeleitete Klasse)
   BitmapMetadata bitmapMetadata = bitmapSource.Metadata as BitmapMetadata;
   if (bitmapMetadata != null)
   {
      Console.WriteLine("Autor(en):");
      if (bitmapMetadata.Author != null)
      {
         foreach (var author in bitmapMetadata.Author)
         {
            Console.WriteLine("   " + author);
         }
      }
      Console.WriteLine("Titel: " + bitmapMetadata.Title);
      Console.WriteLine("Copyright: " + bitmapMetadata.Copyright);
      Console.WriteLine("Bild-Format :" + bitmapMetadata.Format);
      Console.WriteLine("Themenbeschreibung: " + bitmapMetadata.Subject);
      Console.WriteLine("Kommentar: " + bitmapMetadata.Comment);
      Console.WriteLine("Aufnahmedatum: " + bitmapMetadata.DateTaken);
      Console.WriteLine("Kamera-Hersteller: " +
         bitmapMetadata.CameraManufacturer);
      Console.WriteLine("Kamera-Modell: " + bitmapMetadata.CameraModel);
      Console.WriteLine("Name der Anwendung, mit der das Bild " +
         "erzeugt wurde: " + bitmapMetadata.ApplicationName);
```

Listing 627: Ermitteln von Metadaten in einer WPF-Anwendung

```
      Console.WriteLine("Schlüsselwörter:");
      if (bitmapMetadata.Keywords != null)
      {
         foreach (var keyword in bitmapMetadata.Keywords)
         {
            Console.WriteLine(keyword);
         }
      }
      Console.WriteLine("Bildbewertung: " + bitmapMetadata.Rating);   }
   else
   {
      // Unbekannte (neue) Metadaten-Klasse
      Console.WriteLine("Unbekannte (neue) Metadaten-Klasse '" +
         bitmapSource.Metadata.GetType().Name + "'");
   }
}
else
{
   // Keine Metadaten vorhanden
   Console.WriteLine("Keine Metadaten vorhanden");
}
```

Listing 627: Ermitteln von Metadaten in einer WPF-Anwendung (Forts.)

Die Metadaten sind natürlich nicht immer gefüllt. Die Eigenschaften `DateTaken`, `CameraManufacturer` und `CameraModel` sind z. B. normalerweise nur dann mit einem Wert belegt, wenn das Bild von einer Digitalkamera erzeugt wurde.

Windows.Forms

In einer `Windows.Forms`-Anwendung werten Sie Bild-Metadaten leider nicht ganz so elegant wie in WPF über die Eigenschaft `PropertyItems` der `Bitmap`-Klasse aus. Diese Auflistung verwaltet Instanzen der Klasse `PropertyItem`. Die Eigenschaft `Id` gibt als `int`-Wert an, um welche Information es sich handelt. Die Konstanten für die verschiedenen möglichen Id-Werte finden Sie bei Microsoft in der GDI+-Referenz: *msdn.microsoft.com/en-us/library/ms534413(VS.85).aspx*.

Statt die Adresse einzugeben können Sie auch auf der Seite *msdn.microsoft.com/library* einfach nach »Image Property Tag Constants« suchen. Die Beschreibungen der einzelnen Tag-Werte finden Sie über den Link PROPERTY ITEM DESCRIPTIONS.

In der Eigenschaft `Value` wird der Wert der Information gespeichert, eigenartigerweise leider nicht als `object`, sondern als (nicht weiter dokumentiertes) Byte-Array. Die Eigenschaft `Len` gibt die Länge dieses Arrays an. Der Typ des Werts wird in der Eigenschaft `Type` als `short`-Wert verwaltet. Die verwendeten Typen und deren Konstanten finden Sie auf der Seite *msdn.microsoft.com/en-us/library/ms534414(VS.85).aspx*.

Die gespeicherten Werte entsprechen scheinbar (leider nicht dokumentiert) dem EXIF-Standard.

Für Bildinformationen, die als String dargestellt werden können (Textinformationen, Datumswerte), verwaltet die `Value`-Eigenschaft eines `PropertyItem`-Objekts ein (dem EXIF-Standard

entsprechend) nullterminierte (C++-) 8-Bit-Zeichenkette. Die Auswertung solcher Werte ist relativ einfach.

Die Methode `GetTagValueAsString` (Listing 628) liefert den String zurück, der in der Eigenschaft `Value` eines Informations-Werts gespeichert ist. Da es sich bei der dazu auszulesenden Eigenschaft `PropertyItems` leider nur um ein Array handelt, muss diese Methode das Array durchgehen um den gesuchten Wert zu finden. Wurde der Wert gefunden, erfolgt die Konvertierung in einen String einfach über das Durchgehen des Byte-Arrays, wobei das letzte Zeichen (das abschließende 0-Zeichen) nicht mit eingelesen wird.

Datumswerte werden nach dem EXIF-Standard (und im Test bei den von mir getesteten JPEG-Bildern, die von fünf verschiedenen Kameras aufgenommen wurden) im Format *yyyy:MM:dd HH:mm:ss* zurückgegeben (das in der .NET-Dokumentation leider nicht dokumentiert ist). Als Sonderform kommt der String »0000:00:00 00:00:00« vor, der wohl für »kein Datum« steht. Außerdem kann es sein, dass bei einstelligen Zahlen statt der führenden Null ein Leerzeichen steht. Die Methode `GetTagValueAsDateTime` versucht deshalb, den aus einer Tag-Information ausgelesenen String entsprechend in einen `DateTime`-Wert zu konvertieren.

Das Beispiel liest (in einer Konsolenanwendung) alle Bilder eines Ordners und zu jedem Bild den Hersteller und das Modell des Geräts aus, über das das Bild erzeugt wurde, und das Datum der Aufzeichnung des Bilds.

Zum Kompilieren des Beispiels müssen Sie die Namensräume `System`, `System.IO`, `System.Text`, `System.Drawing` und `System.Drawing.Imaging` einbinden.

```
/* Schreibt ausgewählte Bild-Metadaten an die Konsole */
private static void WriteImageMetadata(Bitmap bitmap)
{
   // Hersteller und Name des Geräts ermitteln, über den das Bild
   // erzeugt wurde
   const int PROPERTYTAGEQUIPMAKE = 0x010F;
   const int PROPERTYTAGEQUIPMODEL = 0x110;
   string equipmentManufacturer = GetTagValueAsString(bitmap,
      PROPERTYTAGEQUIPMAKE);
   string equipmentModel = GetTagValueAsString(bitmap, PROPERTYTAGEQUIPMODEL);
   Console.WriteLine("Hersteller, Modell: {0} {1}", equipmentManufacturer,
      equipmentModel);

   // Datum der Erzeugung des Bilds ermitteln
   const int PROPERTYTAGDATETIME = 0x0132;
   DateTime? imageCreateDate = GetTagValueAsDateTime(bitmap,
      PROPERTYTAGDATETIME);
   if (imageCreateDate != null)
   {
      Console.WriteLine("Bild erzeugt am {0}", imageCreateDate.ToString());
   }
```

Listing 628: Methoden zur Ermittlung und Ausgabe des Herstellers, Kameramodells und des Aufnahmedatums von Jpeg-Bildern an der Konsole

```
}

/* Liefert den Wert einer Tag-Eigenschaft eines Bilds als String */
private static string GetTagValueAsString(Bitmap bitmap, int itemType)
{
   string result = null;
   for (int i = 0; i < bitmap.PropertyItems.Length; i++)
   {
      PropertyItem item = bitmap.PropertyItems[i];
      if (item.Id == itemType)
      {
         for (int j = 0; j < item.Len - 1; j++)
         {
            result += (char)item.Value[j];
         }
         break;
      }
   }
   return result;
}

/* Liefert den Wert einer Tag-Eigenschaft eines Bilds als DateTime */
private static DateTime? GetTagValueAsDateTime(Bitmap bitmap, int itemType)
{
   string propertyValue = GetTagValueAsString(bitmap, itemType);
   if (propertyValue != null)
   {
      // Versuch, den im Format yyyy:MM:dd HH:mm:ss
      // ermittelten String in ein Datum zu konvertieren
      if (propertyValue != "0000:00:00 00:00:00")
      {
         // Berücksichtigen, dass bei einstelligen Zahlen auch Leerzeichen
         // statt der 0 angegeben sein können.
         propertyValue = Regex.Replace(propertyValue, @": (\d)", @":0${1}");
         propertyValue = Regex.Replace(propertyValue, @" (\d):", @"0${1}:");

         // Den String versuchen zu parsen
         DateTime result;
         if (DateTime.TryParseExact(propertyValue, "yyyy:MM:dd HH:mm:ss",
            CultureInfo.InvariantCulture, DateTimeStyles.None, out result))
         {
            return result;
         }
         else
         {
```

Listing 628: Methoden zur Ermittlung und Ausgabe des Herstellers, Kameramodells und des Aufnahmedatums von Jpeg-Bildern an der Konsole (Forts.)

```
            throw new Exception("Der String '" + propertyValue +
               "' kann nicht in einen DateTime-Wert " +
               "konvertiert werden");
         }
      }
   }
   return null;
}
```

Listing 628: Methoden zur Ermittlung und Ausgabe des Herstellers, Kameramodells und des Aufnahmedatums von Jpeg-Bildern an der Konsole (Forts.)

Bei Informationen, die als Zahlwert gespeichert sind, wird die Auswertung etwas komplizierter. Das Byte-Array ist in diesem Fall lediglich eine Darstellung der für die Zahlwerte gespeicherten einzelnen Bytes. Für einen gespeicherten `int`-Wert enthält das Array also die vier Bytes, in denen der `int`-Wert gespeichert ist. Einige Zahlwerte werden auch als rationale Zahl (Bruchzahl) verwaltet. Hier wird dann eigentlich ein Array aus zwei Zahlen vom entsprechenden Typ (z. B. `long`) verwaltet, wobei die erste der Zähler und die zweite der Nenner ist. Das Byte-Array enthält dann wieder lediglich die einzelnen Bytes dieser Zahlwerte.

In C++ ist die Auswertung von Zahlwerten ganz einfach. Dazu casten Sie das erste Byte des Byte-Arrays einfach in einen Zeiger des entsprechenden Typs und werten die Zahl dann über den Zeiger aus. Einen `long`-Wert können Sie in C++ dann z. B. so auslesen:

```
long* ptrLong = (long*)(pProp.value);
printf("Der Wert ist %d.\n", ptrLong[0]);
```

Einen als rationale Zahl dargestellten `long`-Wert können Sie in C++ so auslesen:

```
long* ptrLong = (long*)(pProp.value);
printf("Der Wert ist %d/%d.\n", ptrLong[0], ptrLong[1]);
```

In C# können Sie Zahlwerte auch auf diese Weise auswerten, dazu müssen Sie allerdings mit Zeigern in einem unsicheren Codeblock arbeiten (und damit die ganze Assembly mit der Option Unsafe kompilieren). Einfacher ist es, stattdessen die Methoden der Klasse `BitConverter` zu verwenden, die ein Byte-Array in den gewünschten Typ umwandeln.

Als Beispiel habe ich die Helligkeit eines Bilds gewählt, die als `PropertyTagTypeSRational`-Wert gespeichert ist. Ein solcher Wert soll laut der Dokumentation zwei `long`-Werte mit Vorzeichen verwalten, die eine rationale Zahl darstellen. Der erste ist der Nenner und der zweite der Zähler.

In meinen Tests wurden hier aber nicht zwei `long`- sondern zwei `int`-Werte gespeichert (was auch daran zu erkennen ist, dass das Byte-Array lediglich acht Bytes verwaltet). Daran erkennen Sie, dass Sie etwas vorsichtig mit der Dokumentation umgehen und selbst ausprobieren müssen. Zur Sicherheit fragt die folgende Methode zur Ermittlung der Helligkeit eines Bilds die Länge des Arrays ab und ermittelt das Ergebnis entsprechend:

```
private static double GetBitmapBrightness(Bitmap bitmap)
{
   const int PropertyTagExifBrightness = 0x9203;
   double result = 0;
   for (int i = 0; i < bitmap.PropertyItems.Length; i++)
   {
      PropertyItem item = bitmap.PropertyItems[i];
      if (item.Id == PropertyTagExifBrightness)
      {
         // Die Werte für den Zähler (Numerator) und den
         // Nenner (Denominator) ermitteln
         if (item.Len == 8)
         {
            // Zwei int-Werte für den Zähler und den Nenner
            // Anmerkung: Entspricht nicht der Dokumentation,
            // kam in meinen Tests aber ausschließlich vor
            int numerator = BitConverter.ToInt32(item.Value, 0);
            int denominator = BitConverter.ToInt32(item.Value, 4);

            // Das Ergebnis berechnen
            result = numerator / (double)denominator;
         }
         else if (item.Len == 16)
         {
            // Zwei long-Werte für den Zähler und den Nenner
            // Anmerkung: Kam im Test nicht vor, ist aber laut der
            // Dokumentation die korrekte Variante
            long numerator = BitConverter.ToInt32(item.Value, 0);
            long denominator = BitConverter.ToInt32(item.Value, 4);

            // Das Ergebnis berechnen
            result = numerator / (double)denominator;
         }
         break;
      }
   }
   return result;
}
```

Listing 629: Methode zur Ermittlung der Helligkeit eines Bilds

321 Das Aufnahmedatum eines Bilds auslesen

Basierend auf dem Wissen aus Rezept 320 habe ich eine Methode entwickelt, die das Aufnahmedatum eines Bilds ausliest. Diese Methode liegt in zwei Varianten vor, einer für WPF- und einer für `Windows.Forms`-Anwendungen.

Hinweis

Beide Varianten sind nicht 100-prozentig sicher, da das als String zurückgegebene Datum theoretisch jedes Format besitzen kann. Ein Test mit allen meinen Digitalkamera-Bildern (die mit verschiedenen Kameras aufgenommen wurden) hat aber ergeben, dass das Format für WPF immer das Format war, das der aktuellen Kultur entspricht. Für die `Bitmap`-Klasse (die in `Windows.Forms`-Anwendungen verwendet wird) wurde immer das (EXIF-)Format *yyyy:MM:dd HH:mm:ss* zurückgebeben. Die Methoden sollten also für die meisten (Digitalkamera-)Bilder funktionieren. Das Beispiel zu diesem Rezept liest übrigens das Erstelldatum aller Bilder ab einem anzugebenden Ordner ein und hält (mit `Debugger.Break()`) an, wenn ein Datum nicht dem erwarteten Format entspricht. Mit dieser Anwendung können Sie sehr gut prüfen, ob Ihre Bilder ggf. ein anderes Datumsformat verwalten.

Zum Kompilieren der WPF-Variante müssen Sie die Namensräume `System` und `System.Windows.Media.Imaging` importieren. Diese Methode gibt (wie auch die GDI-Variante) eine `Nullable<DateTime>`-Instanz zurück. `null` wird zurückgegeben wenn kein Datum existiert. Im Falle eines Datums, das nicht geparst werden kann, wirft `GetCreationDate` eine `FormatException`.

```
public static DateTime? GetCreationDate(BitmapSource bitmapSource)
{
   // Ermitteln, ob Metadaten vorhanden sind
   if (bitmapSource.Metadata != null)
   {
      // Ermitteln, ob die Metadaten vom Typ BitmapMetadata sind
      // (die einzige zurzeit von ImageMetadata abgeleitete Klasse)
      BitmapMetadata bitmapMetadata = bitmapSource.Metadata as BitmapMetadata;
      if (bitmapMetadata != null)
      {
         if (bitmapMetadata.DateTaken != null)
         {
            // Versuch, das Datum mit der aktuellen Kultur zu konvertieren
            DateTime result;
            if (DateTime.TryParse(bitmapMetadata.DateTaken, out result))
            {
               return result;
            }
            else
            {
               throw new FormatException("Der String '" +
                  bitmapMetadata.DateTaken +
                  "' kann nicht in einen DateTime-Wert " +
                  "konvertiert werden");
            }
         }
      }
```

Listing 630: WPF-Variante der Methode zum Auslesen des Aufnahmedatums eines Bilds

```
   }

   // Das Datum existiert nicht oder ist nicht angegeben
   return null;
}
```

Listing 630: WPF-Variante der Methode zum Auslesen des Aufnahmedatums eines Bilds (Forts.)

Die (langsame) GDI-Variante (die wohl vorwiegend in `Windows.Forms`-Anwendungen Verwendung finden wird, die aus Speicher-Gründen die WPF-Assemblys nicht referenzieren sollen) benötigt den Import der Assemblys `System`, `System.Drawing`, `System.Drawing.Imaging`, `System.Globalization` und `System.Text.RegularExpressions`.

```
public static DateTime? GetCreationDate(Bitmap bitmap)
{
   const int propertyTagDateTime = 0x0132;

   string propertyValue = null;
   for (int i = 0; i < bitmap.PropertyItems.Length; i++)
   {
      PropertyItem item = bitmap.PropertyItems[i];
      if (item.Id == propertyTagDateTime)
      {
         // Den String ermitteln, der das Datum speichert
         for (int j = 0; j < item.Len - 1; j++)
         {
            propertyValue += (char)item.Value[j];
         }

         // Berücksichtigen, dass bei einstelligen Zahlen auch
         // Leerzeichen statt der 0 angegeben sein können.
         propertyValue = Regex.Replace(propertyValue, @": (\d)", @":0${1}");
         propertyValue = Regex.Replace(propertyValue, @" (\d):", @"0${1}:");

         // Versuch, den im Format yyyy:MM:dd HH:mm:ss
         // ermittelten String in ein Datum zu konvertieren
         if (propertyValue != "0000:00:00 00:00:00")
         {
            DateTime result;
            if (DateTime.TryParseExact(propertyValue, "yyyy:MM:dd HH:mm:ss",
               CultureInfo.InvariantCulture, DateTimeStyles.AllowInnerWhite,
               out result))
            {
               return result;
            }
```

Listing 631: GDI-Variante der Methode zum Auslesen des Aufnahmedatums eines Bilds

```
            else
            {
               throw new FormatException("Der String '" + propertyValue +
                  "' kann nicht in einen DateTime-Wert " +
                  "konvertiert werden");
            }
         }

         break;
      }
   }

   // Das Datum existiert nicht oder ist nicht angegeben
   return null;
}
```

Listing 631: GDI-Variante der Methode zum Auslesen des Aufnahmedatums eines Bilds (Forts.)

322 Eingelesene Bilder im Originalformat speichern

Wenn eine Anwendung mehrere beliebige Bilder einliest, diese bearbeitet und wieder speichert, ist es u. U. notwendig, dass die Bilder im Originalformat gespeichert werden. In WPF-Anwendungen benötigen Sie dazu den Decoder, der das Bild beim Einlesen decodiert hat. Wenn Sie mit GDI+ arbeiten (in einer `Windows.Forms`-Anwendung), wird beim Speichern normalerweise das Originalformat verwendet, wenn Sie das Bild aber bearbeitet haben, müssen Sie selbst dafür sorgen, dass das Originalformat zur Verfügung steht.

WPF

In einer WPF-Anwendung lesen Sie Bilder (mehr oder weniger direkt) immer über Instanzen der Klasse `BitmapDecoder` oder davon abgeleitete, auf bestimmte Codecs spezialisierte Klassen wie z. B. `JpegBitmapDecoder`. Wenn Sie `BitmapDecoder` verwenden, ermittelt die `Create`-Methode beim Einlesen eines Bilds den Codec und sucht diesen in der Windows-Registry. Wird ein passender Codec gefunden, wird das Bild über diesen dekodiert und eingelesen. Informationen über den Codec stehen in der Eigenschaft `CodecInfo` zur Verfügung.

Speichern können Sie über Instanzen der `BitmapEncoder`-Klasse oder davon abgeleiteter Klassen (wie `JpegBitmapEncoder`). Die spezialisierten Klassen erwarten keine weitere Angabe (des Codec). Ein Bild in einem der Standardformate zu speichern ist also kein Problem. Die `Create`-Methode der allgemeinen `BitmapEncoder`-Klasse erwartet aber den GUID des Codec, der verwendet werden soll. Und das ist auch schon der Trick um ein Bild in dem Format abzuspeichern, in dem es eingelesen wurde. Den Codec-GUID erhalten Sie nämlich über die Eigenschaft `ContainerFormat` des `CodecInfo`-Objekts.

Das folgende Beispiel liest ein Bild ein, erzeugt davon einen Klon (als Demo für eine Bearbeitung) und speichert den Klon dann unter einem anderen Namen, aber im Originalformat. Zum Kompilieren müssen Sie die Namensräume `System`, `System.IO` und `System.Windows.Media.Imaging` einbinden.

```
// Bild einlesen
string sourceFilename = "C:\\Bilder\\Hitchhiker.gif";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(sourceFilename),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];

// Bild bearbeiten (hier nur: Kopie erzeugen)
BitmapSource destBitmapSource = bitmapSource.Clone();

// Bild mit dem Original-Codec speichern
string destFilename = "C:\\Bilder\\Hitchhiker-Klon.gif";
BitmapEncoder encoder = BitmapEncoder.Create(
   decoder.CodecInfo.ContainerFormat);
encoder.Frames.Add(BitmapFrame.Create(destBitmapSource));
using (FileStream fileStream = new FileStream(destFilename, FileMode.Create))
{
   encoder.Save(fileStream);
}
```

Listing 632: Einlesen, Bearbeiten und Speichern eines Bilds im Originalformat

Windows.Forms

In GDI+ verwaltet ein `Bitmap`-Objekt in seiner Eigenschaft `RawFormat` (siehe Rezept 319) das Bildformat, mit dem das Objekt ursprünglich erzeugt wurde. Ein aus einer Datei eingelesenes `Bitmap`-Objekt verwaltet in dieser Eigenschaft das Format des Quellbilds. Ein im Programm dynamisch erzeugtes `Bitmap`-Objekt besitzt das Format `ImageFormat.MemoryBmp`.

Wenn Sie ein `Bitmap`-Objekt in einer Datei speichern, resultiert eine Datei in dem Format, das die Eigenschaft `RawFormat` angibt. Bei `Bitmap`-Objekten im Format `ImageFormat.MemoryBmp` wird allerdings eine Datei im PNG-Format erzeugt. Wenn Sie nun ein Bild aus einer Datei einlesen und das Bild im Programm bearbeiten, indem Sie ein neues `Bitmap`-Objekt erzeugen und die Daten des eingelesenen Bilds (bearbeitet) in dieses hineinschreiben, resultiert ein `Bitmap`-Objekt in dem Format `ImageFormat.MemoryBmp`. Speichern Sie dieses Objekt in einer Datei, erhalten Sie eine PNG-Datei, unabhängig von der Dateiendung, die Sie angeben:

```
string sourceFilename = "C:\\Bilder\\Hitchhiker.gif";
string destFilename = "C:\\Bilder\\Hitchhiker-Klon.gif";
using (Bitmap sourceBitmap = new Bitmap(sourceFilename))
{
   // Bild bearbeiten (hier nur: Kopie erzeugen)
   using (Bitmap destBitmap = new Bitmap(sourceBitmap))
   {
```

Listing 633: Fehlerhaftes Bearbeiten einer Bilddatei unter GDI+

```
      // Bild speichern, ohne das Format anzugeben
      destBitmap.Save(destFilename);
   }
}
```

Listing 633: Fehlerhaftes Bearbeiten einer Bilddatei unter GDI+ (Forts.)

In diesem Beispiel wird die Bilddatei im PNG-Format gespeichert, obwohl die Quelldatei im GIF-Format vorliegt und als Dateiendung für die Zieldatei *.gif* angegeben ist.

Würden Sie keine Kopie erzeugen (und z. B. die einzelnen Pixel des originalen `Bitmap`-Objekts bearbeiten, wie ich es in Rezept 336 zeige), würde die Datei im GIF-Format gespeichert werden, da `RawFormat.Guid` die GUID des GIF-Formats verwaltet.

In den meisten Fällen ist es beim Bearbeiten von Bildern jedoch notwendig, eine Kopie zu erzeugen. In diesem Fall geben Sie beim Speichern einfach am zweiten Argument das Format des originalen Bilds an:

```
string sourceFilename = "C:\\Bilder\\Hitchhiker.gif";
string destFilename = "C:\\Bilder\\Hitchhiker-Klon.gif";
using (Bitmap sourceBitmap = new Bitmap(sourceFilename))
{
   // Bild bearbeiten (hier nur: Kopie erzeugen)
   using (Bitmap destBitmap = new Bitmap(sourceBitmap))
   {
      // Bild im Format des Quellbilds speichern
      destBitmap.Save(destFilename, sourceBitmap.RawFormat);
   }
}
```

Listing 634: Korrektes Bearbeiten einer Bilddatei unter GDI+

323 Bild in Byte-Array umwandeln

Für einige spezielle Aufgaben wie z. B. das Speichern eines Bilds in einer Datenbank muss das in einem `BitmapSource`-, `Bitmap`-, `Icon`- oder `Metafile`-Objekt gespeicherte Bild in ein Byte-Array umgewandelt werden. Die Klassen dieser Objekte besitzen leider keine Methode `CopyTo` o. Ä., die diese Aufgabe erledigt.

WPF

In WPF können Sie theoretisch die `CopyPixels`-Methode der `BitmapSource`-Klasse verwenden, um ein Bild in ein Byte-Array umzuwandeln. `CopyPixels` erzeugt ein Array, in dem die rohen Pixeldaten des Bilds enthalten sind. Wenn Sie die Daten eines Bilds aber in einer Datenbank speichern wollen, hat dieses Vorgehen (das ich im Rezept 336 auf Seite 973 einsetze) aber den Nachteil, dass Sie sich zum späteren Decodieren mehrere Informationen merken müssen (die Schrittweite des Bilds, die Farbtiefe, die Breite und die Höhe).

Deshalb habe ich eine andere Lösung entwickelt. Diese Lösung verwendet einen `BitmapEncoder`, der die Daten eines Bilds codiert und in einen Stream schreiben kann. Als Stream verwende ich einen `MemoryStream`, der dann einfach über `ToArray` in ein Byte-Array überführt wird.

Der Methode `BitmapSource2Byte` in Listing 635 müssen Sie deswegen neben dem zu konvertierenden Bild auch den `BitmapEncoder` übergeben. Damit überlasse ich Ihnen die Wahl des Bildformats (ich würde das verlustfreie PNG-Format bevorzugen, das allerdings keine Animationen unterstützt ...).

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.IO` und `System.Windows.Media.Imaging` importieren.

```
public static byte[] BitmapSource2Byte(BitmapSource bitmapSource,
   BitmapEncoder encoder)
{
   if (encoder == null)
   {
      throw new ArgumentNullException("encoder");
   }

   // Das Bild dem Encoder hinzufügen
   encoder.Frames.Add(BitmapFrame.Create(bitmapSource));

   // MemoryStream erzeugen und das Bild in diesen schreiben
   using (MemoryStream imageStream = new MemoryStream())
   {
      encoder.Save(imageStream);
      imageStream.Flush();

      // MemoryStream in ein Byte-Array schreiben und dieses zurückgeben
      return imageStream.ToArray();
   }
}
```

Listing 635: Methode zum Umwandeln eines BitmapSource-Objekts in ein Byte-Array

Die Anwendung dieser Methode ist einfach. Das folgende Beispiel verwendet den `PngBitmapEncoder` zum Kodieren der Bilddaten im PNG-Format:

```
// Die Datei in ein BitmapSource-Objekt einlesen
BitmapSource bitmapSource = BitmapFrame.Create(new Uri(imageFilename));
// Das Bitmap in ein byte-Array umwandeln
byte[] imageData2 = ImageUtils.BitmapSource2Byte(bitmapSource,
   new PngBitmapEncoder());
```

Listing 636: Beispielhafte Anwendung der BitmapSource2Byte-Methode

Windows.Forms

In GDI+ ist die `Save`-Methode der Basisklasse `Image` in der Lage, die Bilddaten in einen Stream zu schreiben, wobei das Bildformat angegeben werden kann. Wenn Sie am ersten Argument einen `MemoryStream` übergeben, können Sie aus diesem nach dem Aufruf von `Save` ein Byte-Array erzeugen, wie es die Methode `Image2Byte` in Listing 637 zeigt. Am zweiten Argument der `Save`-Methode wird das Format übergeben, in dem die »Datei« geschrieben werden soll (Gif, Jpeg etc.). Um dieses nicht festzulegen erwartet `Image2Byte` das Format am zweiten Argument.

`Image2Byte` benötigt die Einbindung der Namensräume `System.Drawing`, `System.Drawing.Imaging` und `System.IO`.

```
public static byte[] Image2Byte(Image image, ImageFormat format)
{
   // MemoryStream erzeugen und das Bild in diesen schreiben
   MemoryStream imageStream = new MemoryStream();
   image.Save(imageStream, format);
   imageStream.Flush();

   // MemoryStream in ein Byte-Array schreiben und dieses zurückgeben
   return imageStream.ToArray();
}
```

Listing 637: GDI-Methode zum Schreiben eines Image-Objekts in ein Byte-Array

324 Byte-Array in Bild umwandeln

Wenn Sie Bilder als Byte-Array z. B. in einer Datenbank speichern (Rezept 323), müssen Sie diese auch wieder decodieren. Und dabei hilft dieses Rezept.

WPF

Wenn Sie ein Bild nach dem Rezept 323 in WPF in ein Byte-Array geschrieben haben, das einem bestimmten Bildformat entspricht, können Sie dieses recht einfach wieder in ein `BitmapSource`-Objekt konvertieren, indem Sie die Byte-Daten in einen `MemoryStream` schreiben und diesen der `Create`-Methode der `BitmapDecoder`-Klasse übergeben (die das Format an den Daten automatisch erkennt). Die Methode `Byte2BitmapSource` in Listing 638 macht genau das. Damit der `MemoryStream` sauber geschlossen wird, erzeugt diese Methode den Stream in einer `using`-Anweisung. Der `BitmapDecoder` wird mit der `BitmapCacheOption OnLoad` erzeugt, was sehr wichtig ist, denn diese Option sorgt dafür, dass die Bitmap-Daten sofort aus dem Stream eingelesen werden (und nicht erst, wenn das Bild benötigt wird, das dann zu spät ist, da der Stream zwischenzeitlich geschlossen wurde).

`Byte2BitmapSource` benötigt den Import der Namensräume `System.IO` und `System.Windows.Media.Imaging`.

```
public static BitmapSource Byte2BitmapSource(byte[] imageBytes)
{
   // MemoryStream mit den Bytes des Bilds erzeugen und
   // ein damit erzeugtes Bitmap zurückgeben
   using (MemoryStream imageStream = new MemoryStream(imageBytes))
   {
      BitmapDecoder decoder = BitmapDecoder.Create(imageStream,
         BitmapCreateOptions.None, BitmapCacheOption.OnLoad);
      return decoder.Frames[0];
   }
}
```

Listing 638: WPF-Methode zum Erzeugen eines BitmapSource-Objekts aus einem Byte-Array

Hinweis

Das Beispiel zu diesem Rezept beweist, dass das Konvertieren in ein Byte-Array und das Decodieren für alle gängigen Bildformate problemlos funktioniert. Schauen Sie es sich an ...

Windows.Forms

In `Windows.Forms`-Anwendungen können Sie ein als Byte-Array gespeichertes Bitmap wie bei WPF über einen `MemoryStream` in ein Bild umwandeln. Der Vergleich mit WPF ergab hier sogar einen kleinen Performance-Vorteil (siehe Beispiel zu diesem Rezept).

Die Methode `Byte2Bitmap` in Listing 639 zeigt, wie das geht. Um das `Bitmap`-Objekt zu erzeugen, liest diese Methode das Byte-Array in eine neue `MemoryStream`-Instanz ein und übergibt diese an den Konstruktor der `Bitmap`-Klasse.

`Byte2Bitmap` erfordert die Referenzierung der Assembly *System.Drawing.dll* und das Importieren der Namensräume `System.Drawing` und `System.IO`.

```
public static Bitmap Byte2Bitmap(byte[] imageBytes)
{
   // MemoryStream mit den Bytes des Bilds erzeugen und
   // ein damit erzeugtes Bitmap zurückgeben
   using (MemoryStream imageStream = new MemoryStream(imageBytes))
   {
      return new Bitmap(imageStream);
   }
}
```

Listing 639: GDI- Methode zum Erzeugen eines Bitmap-Objekts aus einem Byte-Array

325 Bilder aus der Zwischenablage auslesen

Ein Bild, das in der Zwischenablage gespeichert ist, können Sie in WPF und in `Windows.Forms` auslesen.

WPF

In WPF können Sie theoretisch die `GetImage`-Methode der Klasse `Clipboard` (aus dem Namensraum `System.Windows`) verwenden um ein Bild aus der Zwischenablage einzulesen. Über die `ContainsImage`-Methode können Sie überprüfen, ob überhaupt ein Bild in der Zwischenablage gespeichert ist. `GetImage` gibt allerdings einfach `null` zurück, wenn die Zwischenablage kein Bild beinhaltet.

```
if (Clipboard.ContainsImage)
{
   BitmapSource bitmapSource = Clipboard.GetImage();
   ...
}
```

Listing 640: (Nicht so ganz funktionierendes) Einlesen eines Bilds aus der Zwischenablage in WPF

So weit die Theorie. Leider scheint das Ganze nicht so wirklich gut zu funktionieren. Sie können ein so eingelesenes Bild zwar problemlos in eine Datei speichern. Wenn Sie das Bild aber direkt nach dem Einlesen der `Source`-Eigenschaft eines `Image`-Steuerelements zuweisen, wird das Bild aus unerfindlichen Gründen entweder gar nicht angezeigt (bei kleinen Bildern, die z. B. aus Paint heraus kopiert wurden) oder nur zerstückelt (wenn Sie z. B. mit der Druck-Taste den gesamten Desktop in die Zwischenablage kopieren). Vergleichen Sie dazu auch den Blog-Eintrag *shevaspace.spaces.live.com/Blog/cns!FD9A0F1F8DD06954!441.entry*.

Deshalb gehe ich mit meiner Lösung den (funktionierenden) Weg über GDI+. Die folgende Methode, die in ähnlicher Form auch für das Auslesen von Bildern in `Windows.Forms`-Anwendungen verwendet wird, setzt die `Clipboard`-Klasse aus `Windows.Forms` ein, um ein `Bitmap`-Objekt aus der Zwischenablage einzulesen, und konvertiert dieses schließlich in ein `BitmapSource`-Objekt.

`GetBitmapSourceFromClipboard` benötigt den Import der Namensräume `System`, `System.Windows` und `System.Windows.Media.Imaging`.


```
public static BitmapSource GetBitmapSourceFromClipboard()
{
   // Die Zwischenablagedaten auslesen und überprüfen
   System.Windows.Forms.IDataObject clipboardData =
      System.Windows.Forms.Clipboard.GetDataObject();
   if (clipboardData != null)
   {
```

Listing 641: (Funktionierendes) Einlesen eines Bilds aus der Zwischenablage in WPF (über den Umweg über GDI)

```
      // Überprüfen, ob ein Bitmap gespeichert ist
      if (clipboardData.GetDataPresent(
         System.Windows.Forms.DataFormats.Bitmap))
      {
         // Bitmap auslesen
         System.Drawing.Bitmap bitmap =
            (System.Drawing.Bitmap)clipboardData.GetData(
            System.Windows.Forms.DataFormats.Bitmap);

         // Das Bitmap in ein BitmapSource-Objekt umwandeln
         return System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap(
           bitmap.GetHbitmap(), IntPtr.Zero, Int32Rect.Empty,
           BitmapSizeOptions.FromEmptyOptions());
      }
   }

   // null zurückgeben, falls in der Zwischenablage kein Bild
   // gespeichert ist
   return null;
}
```

Listing 641: (Funktionierendes) Einlesen eines Bilds aus der Zwischenablage in WPF (über den Umweg über GDI) (Forts.)

Windows.Forms

In `Windows.Forms`-Anwendungen können Sie Bilder aus der Zwischenablage auslesen, indem Sie die `GetData`-Methode der `Clipboard`-Klasse (aus dem Namensraum `System.Windows.Forms`) aufrufen und als Format-Typ den Typ der `Bitmap`-Klasse übergeben. Zuvor sollten Sie überprüfen, ob überhaupt ein Bitmap-Bild in der Zwischenablage gespeichert ist, wozu Sie einen Vergleich der Daten auf `null` und die `GetDataPresent`-Methode verwenden können.

Die Methode `GetBitmapFromClipboard` geht auf diese Weise vor um ein in der Zwischenablage gespeichertes `Bitmap`-Bild auszulesen. Enthält die Zwischenablage keine Daten oder kein `Bitmap`-Bild, gibt die Methode `null` zurück.

Zum Kompilieren dieser Methode müssen Sie die Assemblys *System.Windows.Forms.dll* und *System.Drawing.dll* referenzieren und die Namensräume `System.Windows.Forms` und `System.Drawing` einbinden.

```
public static Bitmap GetBitmapFromClipboard()
{
   // Die Zwischenablagedaten auslesen und überprüfen
   IDataObject clipboardData = Clipboard.GetDataObject();
   if (clipboardData != null)
   {
```

Listing 642: Methode zum Auslesen eines Bitmap-Objekts aus der Zwischenablage

```
      // Überprüfen, ob ein Bitmap gespeichert ist
      if (clipboardData.GetDataPresent(typeof(Bitmap)))
      {
         return (Bitmap)clipboardData.GetData(typeof(Bitmap));
      }
   }

   // null zurückgeben, falls kein Bitmap in der Zwischenablage
   // gespeichert ist
   return null;
}
```

Listing 642: Methode zum Auslesen eines Bitmap-Objekts aus der Zwischenablage (Forts.)

326 Screenshot erstellen

Einen Screenshot des Bildschirms bzw. eines Formulars können Sie mit .NET-Klassen erzeugen. WPF stellt dazu, soweit ich herausgefunden habe, allerdings keine Funktionalität zur Verfügung. Sie müssen also, auch unter WPF, auf das systemeigene GDI+ zurückgreifen.

Prinzipiell können Sie einen Screenshot erzeugen, idem Sie ein `Bitmap`-Objekt von der benötigten Größe erzeugen, für dieses ein `Graphics`-Objekt erzeugen und dessen `CopyFromScreen`-Methode aufrufen. `CopyFromScreen` erwartet am ersten und zweiten Argument die X- und Y-Position des zu kopierenden Bildschirmausschnitts, am dritten und vierten Argument die X- und Y-Position im Ziel-Bitmap, an die kopiert werden soll (normalerweise 0, 0), am fünften Argument die Angabe der Größe des zu kopierenden Bildschirmausschnitts und am letzten Argument einen oder mehrere Werte der `CopyPixelOperation`-Aufzählung. Der `CopyPixelOperation`-Wert bestimmt, wie die einzelnen Pixel des Bildschirms auf das Bild kopiert werden. Für einen Screenshot wäre zumindest die Angabe `CopyPixelOperation.SourceCopy` notwendig, damit die Pixel der Quelle 1:1 in das Ziel-Bild kopiert werden:

```
Rectangle screenClip = Screen.PrimaryScreen.Bounds;

// Bitmap für das Ergebnis erzeugen und das& zugehörige
// Graphics-Objekt auslesen
Bitmap bitmap = new Bitmap(screenClip.Width, screenClip.Height);
Graphics g = Graphics.FromImage(bitmap);

// Den Bildschirminhalt in das Bitmap kopieren
g.CopyFromScreen(screenClip.Left, screenClip.Top, 0, 0,
   new Size(screenClip.Width, screenClip.Height),
   CopyPixelOperation.SourceCopy);
```

Listing 643: Prinzipiell mögliches Erstellen eines Screenshot

In WPF können Sie das `Bitmap`-Objekt dann in ein `BitmapSource`-Objekt umwandeln, wie ich es in Rezept 318 zeige.

Das Problem mit dieser Variante ist lediglich, dass sie nicht für (mehr oder weniger) transparente Fenster funktioniert. Wenn Sie z. B. die `Opacity`-Eigenschaft eines `Windows.Forms`-Formulars auf einen Wert kleiner 1 setzen und einen Screenshot von dem Bereich erzeugen, auf dem das Formular liegt, ist das Formular auf dem Screenshot nicht sichtbar.

Um halb-transparente Fenster in den Screenshot zu integrieren wäre die zusätzliche Angabe von `CopyPixelOperation.CaptureBlt` notwendig, was laut Dokumentation dafür sorgt, dass Fenster, die andere überlappen, mit in den Screenshot aufgenommen werden. Leider lässt `CopyFromScreen` aber die Kombination `CopyPixelOperation.SourceCopy | CopyPixelOperation.CaptureBlt` nicht zu und wirft in diesem Fall eine `InvalidEnumArgumentException`.

Die Lösung für das Problem ist, dass Sie das, was `CopyFromScreen` macht, selbst implementieren. Und dazu benötigen Sie API-Funktionen. Und ein wenig Theorie (aber nur ein ganz klein wenig):

Windows verwaltet in einem System mit mehreren angeschlossenen Monitoren alle Bildschirme gemeinsam als einen virtuellen Bildschirm. Wenn Sie z. B. zwei Bildschirme mit einer Auflösung von 1280 * 1024 besitzen und diese in den Eigenschaften des Desktop so platziert haben, dass der sekundäre Monitor rechts vom primären liegt, beginnt der primäre Bildschirm an der X-Position 0 und der sekundäre an der X-Position 1281. Da Sie den sekundären Monitor aber auch links vom primären platzieren können, kann die X-Position eines Bildschirms auch negativ sein. Ähnliches gilt für die Y-Position, die alle möglichen Werte besitzen kann, da Sie die einzelnen Bildschirme frei auf dem virtuellen Bildschirm ausrichten können.

Über die Funktion `CreateDC` können Sie nun den *Device Context* (DC) des virtuellen Bildschirms ermitteln. Ein Device Context repräsentiert im Windows-API die Zeichenoberfläche eines Fensters, Bitmaps oder eines Geräts wie eines Druckers. Über die Funktion `BitBlt` können Sie die Farbinformationen eines DC in einen anderen kopieren. Als Ziel-DC verwenden Sie zur Erstellung eines Screenshots den DC eines neu erzeugten `System.Drawing.Bitmap`-Objekts.

Die Methode `Screenshot` in Listing 645 zeigt, wie Sie diese Technik einsetzen können. Diese Methode erwartet am einzigen Argument ein Rechteck, das den zu kopierenden Bildschirm-Ausschnitt angibt. Dieser Ausschnitt bezieht sich auf den virtuellen Bildschirm.

Zum Kompilieren dieser Methode müssen Sie die Assemblys *System.Drawing.dll* und *System.Windows.Forms.dll* referenzieren und die Namensräume `System`, `System.Drawing`, `System.Windows.Forms` und `System.Runtime.InteropServices` importieren.

Zur Umsetzung sind zunächst einige API-Deklarationen notwendig:

```
[DllImport("gdi32.dll", SetLastError=true)]
private static extern int BitBlt(IntPtr hdcDest, int nXDest, int nYDest,
   int nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc,
   CopyPixelOperation dwRop);

[DllImport("gdi32.dll", SetLastError=true)]
private static extern IntPtr CreateDC(string lpszDriver, string lpszDevice,
```

Listing 644: Deklaration der benötigten API-Funktionen und Konstanten

```
   string lpszOutput, IntPtr lpInitData);

private static int SRCCOPY = 0x00CC0020;
```

Listing 644: Deklaration der benötigten API-Funktionen und Konstanten (Forts.)

`Screenshot` ermittelt zunächst über `CreateDC` den DC des Bildschirms. Am ersten Argument wird dazu der String "DISPLAY" übergeben. Mit diesem DC erzeugt `Screenshot` ein `Graphics`-Objekt, das beim nachfolgenden Erzeugen eines `Bitmap`-Objekts am letzten Argument angegeben wird und das die Auflösung des Bilds bestimmt. Das `Bitmap`-Objekt wird dann in der Größe des Bildschirms erzeugt und ist das Ziel für den späteren `BitBlt`-Aufruf. Um den DC dieses Objekts zu erhalten, erzeugt `Screenshot` ein neues `Graphics`-Objekt und gibt das `Bitmap`-Objekt am Argument des Konstruktors an. Der DC wird dann über die Methode `GetHdc` dieses Objekts ausgelesen.

Vor dem Aufruf von `BitBlt` muss der DC des Bildschirms erneut ermittelt werden, da dieser anscheinend beim Aufruf der `FromHdc`-Methode der `Graphics`-Klasse freigegeben wurde (ohne dieses erneute Auslesen schlägt `BitBlt` ohne Fehlermeldung fehl und die Freigabe des DC führt zu einem Fehler).

Dann wird das Bild über `BitBlt` kopiert. Am letzten Argument übergibt `Screenshot` den Wert `CopyPixelOperation.SourceCopy | CopyPixelOperation.CaptureBlt`, was zum einen dafür sorgt, dass das Quellbild alle Farbinformationen des Ziels überschreibt. Zum anderen ermöglichen Sie damit auch das Kopieren von halbtransparenten Fenstern. Über andere Konstanten, die in der Referenz erläutert werden, können Sie die Farbinformationen des Ziels auch auf verschiedene Weise mit denen der Quelle vermischen, was aber für unsere Lösung nicht interessant ist. Schließlich müssen die DCs noch über die `ReleaseHdc`-Methode des jeweiligen `Graphics`-Objekts freigegeben werden, da diese Freigabe nicht automatisch über den Garbage Collector erfolgt.

```
public static Bitmap GetScreenshot(Rectangle screenClip)
{
   // Device Context für den virtuellen Windows-Bildschirm ermitteln
   // und damit ein Graphics-Objekt erzeugen
   IntPtr screenDC = CreateDC("DISPLAY", null, null, (IntPtr)null);
   using (Graphics screenGraphics = Graphics.FromHdc(screenDC))
   {
      // Bitmap mit den Ausmaßen des zu erzeugenden Ausschnitts
      // und der Auflösung des Graphics-Objekts erzeugen
      Bitmap bitmap = new Bitmap(screenClip.Width, screenClip.Height,
         screenGraphics);

      // Zweites Graphics-Objekt aus dem noch leeren Bitmap erzeugen
      // um den DC des Bitmap-Objekts auslesen zu können
      using (Graphics bitmapGraphics = Graphics.FromImage(bitmap))
```

Listing 645: Methode zum Erzeugen eines Screenshots eines Ausschnitts des virtuellen Bildschirms

```
      {
         IntPtr bitmapDC = bitmapGraphics.GetHdc();

         // DC des Bildschirms noch einmal ermitteln
         screenDC = screenGraphics.GetHdc();

         // Über BitBlt das über den Bildschirm-DC repräsentierte Bild
         // in das über den Bitmap-DC repräsentierte Bild kopieren
         if (BitBlt(bitmapDC, 0, 0, screenClip.Width, screenClip.Height,
            screenDC, screenClip.Left, screenClip.Top,
            CopyPixelOperation.SourceCopy |
            CopyPixelOperation.CaptureBlt) == 0)
         {
            bitmapGraphics.ReleaseHdc(bitmapDC);
            screenGraphics.ReleaseHdc(screenDC);
            throw new Exception("API-Fehler " + Marshal.GetLastWin32Error() +
               " beim Aufruf von BitBlt");
         }

         // Die DCs freigeben und das Bild zurückgeben
         bitmapGraphics.ReleaseHdc(bitmapDC);
         screenGraphics.ReleaseHdc(screenDC);
         return bitmap;
      }
   }
}
```

Listing 645: Methode zum Erzeugen eines Screenshots eines Ausschnitts des virtuellen Bildschirms (Forts.)

Die Anwendung der Methode ist nun sehr einfach. Wenn Sie z. B. einen Screenshot des primären Bildschirms erstellen wollen, geben Sie die `Bounds`-Eigenschaft des `Screen`-Objekts an, das `Screen.PrimaryScreen` referenziert:

```
Bitmap captureBitmap = GetScreenshot(Screen.PrimaryScreen.Bounds);
```

In WPF müssen Sie übrigens wohl leider auch die `Screen`-Klasse aus dem Namensraum `System.Windows.Forms` verwenden, da WPF anscheinend keine Möglichkeit bietet, Informationen über die Bildschirme des Systems auszulesen (außer der Breite und Höhe des primären Bildschirms über `SystemParameters.PrimaryScreenWidth` und `SystemParameters.PrimaryScreenHeight`). Das zurückgegebene `Bitmap` wandeln Sie für WPF-Anwendungen über das Rezept 318 in ein `BitmapSource`-Objekt um.

Wollen Sie einen Screenshot eines (sichtbaren) Formulars erzeugen, geben Sie dessen `Bounds`-Eigenschaft an:

```
Bitmap captureBitmap = GetScreenshot(form.Bounds);
```

In WPF-Anwendungen müssen Sie zur Erstellung eines Screenshots eines Fensters dessen auf den (virtuellen) Bildschirm bezogene Pixel-Position und -Größe ermitteln. Über die Methode

PointToScreen können Sie die geräteunabhängigen 1/96-Zoll-Punkte von WPF in Bildschirm-Koordinaten umrechnen. Dabei werden allerdings der Titel und der Rand des Fensters nicht mit eingerechnet. Diesen müssen Sie also hinzurechnen, was natürlich davon abhängt, ob das Fenster einen Rand und/oder einen Titel besitzt). Leider fehlt WPF-Fenstern eine Information über die Größe des Innenbereichs (wie bei Windows.Forms-Formularen über die ClientRectangle-Eigenschaft). Darüber könnten Sie recht einfach die Breite des Randes und der Titelzeile bestimmen (Gesamtbreite – Innenbereich-Breite, Gesamthöhe – Innenbereich-Höhe). Als Notlösung könnten Sie die Eigenschaften WindowCaptionHeight (Höhe der Fenster-Titelzeile), ThickVerticalBorderWidth (Breite eines breiten rechten und linken Rahmens), ThinVerticalBorderWidth (Breite eines schmalen rechten und linken Rahmens), ThickHorizontalBorderHeight (Höhe eines dünnen oberen und unteren Rahmens) und ThickHorizontalBorderHeight (Höhe eines dünnen oberen und unteren Rahmens) der SystemParameters-Klasse verwenden. Das war mir aber zu kompliziert und unsicher, da Fenster unterschiedliche Rahmen besitzen können.

Meine Lösung des Problems setzt eine eigene Berechnung ein. Dazu müssen Sie allerdings die Auflösung des Bildschirms ermitteln, auf dem das Fenster liegt. Diese ist zwar normalerweise 96 DPI, kann aber auch eine andere sein (was Sie unter Windows in den Eigenschaften des Desktop einstellen können). Also muss ein wenig gerechnet werden. Die Methode GetScreenshot in Listing 646 zeigt, wie dies geht.

```
public static Bitmap GetScreenshot(Window window)
{
   // Das Argument überprüfen
   if (window == null)
   {
      throw new ArgumentNullException("window");
   }

   // PresentationSource für das Visual-Objekt erzeugen,
   // das das übergebene Fenster darstellt
   PresentationSource presentationSource =
      PresentationSource.FromVisual(window);
   if (presentationSource != null)
   {
      // Die Auflösung ermitteln
      double dpiX = 96.0 *
         presentationSource.CompositionTarget.TransformToDevice.M11;
      double dpiY = 96.0 *
         presentationSource.CompositionTarget.TransformToDevice.M22;

      // Die auf den  Bildschirm bezogene Position
      // und Größe des Fensters ermitteln
      int screenLeft = (int)(window.Left * 96 / dpiX);
```

Listing 646: Methode zur Erstellung eines Screenshot für ein WPF-Fenster

```
      int screenTop = (int)(window.Top * 96 / dpiY);
      int screenWidth = (int)(window.Width * 96 / dpiX);
      int screenHeight = (int)(window.Height * 96 / dpiY);
      System.Drawing.Rectangle screenClip = new System.Drawing.Rectangle(
         screenLeft, screenTop, screenWidth, screenHeight);

      // Mit dem auf den Bildschirm bezogenen Rechteck
      // einen Screenshot erzeugen und zurückgeben
      return GetScreenshot(screenClip);
   }
   else
   {
      throw new Exception("Kann für das Fenster kein "
         "PresentationSource-Objekt erzeugen");
   }
}
```

Listing 646: Methode zur Erstellung eines Screenshot für ein WPF-Fenster (Forts.)

Hinweis

Die Beispiele zu diesem Rezept zeigen einmal, wie Sie `GetScreenshot` in einer `Windows.Forms`- und zum anderen in einer WPF-Anwendung einsetzen.

327 Bilder skalieren

In WPF und in GDI+ skalieren Sie Bilder auf verschiedene Weise. WPF hat dabei den Vorteil, dass die Skalierung sehr viel schneller ist: In einem Performancetest, den Sie neben den Beispielen zu diesem Rezept finden, hat das 100fache Skalieren eines Bilds in Schritten von 1% bis 100% der Originalgröße mit den WPF-Klassen lediglich ca. 0,033 Sekunden benötigt. Das Skalieren mit GDI+ benötigte allerdings 7,6 Sekunden (mit Default-Interpolation) bzw. 23 Sekunden (mit eigener Interpolation). GDI+ hat gegenüber WPF den (wahrscheinlich selten genutzten) Vorteil, dass Sie die Interpolation bestimmen können.

WPF

In WPF skalieren Sie Bilder, indem Sie ein neues `TransformedBitmap`-Objekt erzeugen und am Konstruktor neben dem zu transformierenden `BitmapSource`-Objekt eine `ScaleTransform`-Instanz übergeben. `TransformedBitmap` ist eine von `BitmapSource` abgeleitete Klasse, die in ihrem Konstruktor Transformationen auf dem übergebenen Bild ausführt. Neben Skalierungen können Sie auch andere Transformationen wie z. B. (über `RotateTransform`) ein Drehen des Bilds ausführen.

Der zum Skalieren verwendeten `ScaleTransform`-Instanz übergeben Sie am Konstruktor einen Faktor für die neue Breite und die neue Höhe. Das Skalieren gestaltet sich damit sehr einfach:

```
// Bild einlesen
string sourceFilename = "C:\\Bilder\\Karpathos.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(sourceFilename),
   BitmapCreateOptions.None, BitmapCacheOption.Default);
BitmapSource bitmapSource = decoder.Frames[0];

// Skalieren
double scaleFactor = 0.5;
ScaleTransform scaleTransform = new ScaleTransform(scaleFactor, scaleFactor);
TransformedBitmap transformedBitmap = new TransformedBitmap(bitmapSource,
   scaleTransform);
```

Listing 647: Einlesen und Skalieren eines Bilds

Der Beispiel-Quellcode benötigt den Import der Namensräume `System`, `System.Windows.Media` und `System.Windows.Media.Imaging`.

Die Qualität der Skalierung ist sehr gut. Leider können Sie aber in WPF die Interpolation nicht selbst bestimmen. In einigen Fällen ergibt eine andere Interpolation als die, die `ScaleTransform` verwendet (welche das ist, konnte ich nicht herausfinden), ein besseres Ergebnis (wie Sie ggf. mit dem GDI-Beispiel nachvollziehen können).

Windows.Forms

In Windows.Forms können Sie Bilder skalieren, indem Sie mit einer Instanz einer von `Image` abgeleiteten Klasse (`Bitmap`, `Icon`, `Metafile`) ein neues Objekt erzeugen. Dazu übergeben Sie das Quell-Objekt am ersten Argument des Konstruktors. Am zweiten Argument übergeben Sie die neue Breite und am dritten die neue Höhe. Listing 648 zeigt dies am Beispiel. Eine eingelesene Bitmap-Datei wird in diesem Beispiel auf 50 Prozent der Größe skaliert und unter einem neuen Namen abgespeichert. Zum Kompilieren dieses Quellcodes müssen Sie den Namensraum `System.Drawing` importieren.

```
string sourceFileName = @"C:\\Bilder\\Karpathos.jpg";
Bitmap sourceBitmap = new Bitmap(sourceFileName);
float scaleFactor = 0.5;

string destFilename = @"c:\\Hitchhiker_Scaled.bmp";
Bitmap scaledBitmap = new Bitmap(sourceBitmap,
   (int)(sourceBitmap.Width * scaleFactor),
   (int)(sourceBitmap.Height * scaleFactor));
scaledBitmap.Save(destFilename);
```

Listing 648: Einfaches Skalieren eines Bilds auf eine Größe von 50 Prozent mit GDI+

Die Qualität der Skalierung ist größtenteils erstaunlich gut und hängt davon ab, welches Bildformat Sie skalieren, welche Qualität das Bild aufweist und wie stark Sie hoch- oder herunterskalieren. Beim Skalieren wird eine Interpolation der Bildpunkte durchgeführt. Welche das ist, konnte ich aber nicht ermitteln, da das Ergebnis des einfachen Skalierens mit keinem der

Ergebnisse der möglichen Interpolationen übereinstimmte. Die Interpolation ist nicht immer optimal und kann in vielen Fällen verbessert werden.

Sie können die Interpolation selbst bestimmen, indem Sie das Bild selbst skalieren. Dazu erstellen Sie ein neues `Bitmap`-Objekt in der neuen Größe und erzeugen über `Graphics.FromImage` dazu ein `Graphics`-Objekt. Über dessen `DrawImage`-Methode zeichnen Sie später das originale Bitmap auf das neue, wobei Sie das Quell- und das Ziel-Rechteck in Form von `Rectangle`-Objekten angeben. Die Rechtecke entsprechen beim Skalieren dem Rechteck des Quell- bzw. des Ziel-Bitmaps. Die Interpolation bestimmen Sie über die `InterpolationMode`-Eigenschaft des `Graphics`-Objekts. Über `InterpolationMode.HighQualityBicubic` erreichen Sie (laut der Hilfe in Photoshop) die beste Qualität. Die bikubische Interpolation ist die langsamste, aber genaueste Methode, über die die gleichmäßigsten Tonabstufungen erzeugt werden. `HighQualityBilinear` führt zu einer etwas verminderten, aber dennoch sehr guten Qualität. Für andere, leider nicht näher dokumentierte Qualitätsstufen stehen Ihnen zudem die Konstanten `Bicubic`, `Bilinear`, `NearestNeighbor`, `Default`, `Invalid`, `Low` und `High` zur Verfügung. `High` ergab in meinen Versuchen ebenfalls eine sehr gute Qualität. Eine qualitativ hochwertige Interpolation führt aber natürlich zu einer verminderten Ausführungsgeschwindigkeit.

Da die Einstellung der Interpolation einen großen Einfluss auf die Qualität des Bilds und auf die Geschwindigkeit der Skalierung hat, habe ich die Methode `ScaleBitmap` in Listing 649 so geschrieben, dass Sie die Interpolation bestimmen können. `ScaleBitmap` erzeugt aus dem am ersten Argument übergebenen `Bitmap`-Objekt mit den am zweiten und dritten Argument übergebenen Breiten- und Höhenangaben und den Informationen zur Interpolation ein neues, skaliertes `Bitmap`-Objekt und gibt eine Referenz darauf zurück.

JPEG-Bilder können, wie Sie ja sicherlich wissen, komprimiert gespeichert werden. Die Höhe der Kompression wird auch als Qualität bezeichnet. Eine Qualität von 100 Prozent bedeutet, dass das Bild nicht komprimiert ist. Beim Skalieren von JPEG-Bildern sollte darauf geachtet werden, dass die Qualität des ursprünglichen Bilds in das skalierte Bild übertragen wird, da es ansonsten vorkommen kann, dass das (mit 100 Prozent Qualität gespeicherte) skalierte Bild eine größere Datei ergibt als das originale. Außerdem bringt es keinen Vorteil, ein Bild, das beispielsweise mit einer Qualität von 70 Prozent gespeichert war, mit einer hundertprozentigen Qualität skaliert zu speichern. Es wäre also sinnvoll, die Qualität des Originals in das skalierte Bild zu übernehmen bzw. das skalierte Bild mit der originalen Qualität zu speichern. Dummerweise wird die Qualität nicht direkt in der JPEG-Datei gespeichert. JPEG-Kompressionsverfahren sind recht kompliziert und basieren auf so genannten *Quantisierungstabellen*. Und nur diese sind in der JPEG-Datei gespeichert. Leider existieren mehrere Kompressionsverfahren (z. B. das der *Independent JPEG Group*) und damit auch verschiedene Quantisierungstabellen. Damit ist es unmöglich (oder besser: bisher noch keinem Programmierer gelungen, der sein Wissen veröffentlicht hat) die Qualität auszulesen. Was Sie aber machen können, ist, einfach die Quantisierungstabellen des Originalbilds in das skalierte Bild zu übertragen. Die Quantisierungstabellen erreichen Sie über die Eigenschaft `PropertyItems`, die Instanzen der Klasse `PropertyItem` verwaltet. Wenn Sie diese in einer Schleife durchgehen, können Sie die Tabellen über die `SetPropertyItem`-Methode des skalierten Bitmaps übertragen:

```
foreach (PropertyItem pi in source.PropertyItems)
{
   scaledBitmap.SetPropertyItem(pi);
}
```

Bei Versuchen mit verschiedenen Bilddateien habe ich aber herausgefunden, dass das Kopieren der `PropertyItem`-Objekte zu keinem besseren Ergebnis führt. Eher im Gegenteil waren die resultierenden Dateien bei meinen Versuchen mit JPEG-Bildern sogar noch etwas größer. Die Qualität wurde überhaupt nicht beeinflusst. Außerdem schien bei kopierten `PropertyItem`-Objekten in der Mitte des Bilds ein kleiner Streifen einfach zu fehlen. Deswegen habe ich dieses Kopieren nicht in die Methode `ScaleBitmap` eingebaut.

Zum Kompilieren dieser und der folgenden Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D` importieren.

```
public static Bitmap ScaleBitmap(Bitmap source, int width, int height,
   InterpolationMode interpolationMode)
{
   // Bitmap in der neu berechneten Größe erstellen
   Bitmap scaledBitmap = new Bitmap(width, height);

   // Alle PropertyItem-Objekte des Quellbilds
   // in das neue Bitmap übertragen und damit für
   // JPEG-Bilder deren Quantisierungstabellen
   // (Kompression, Qualität) übernehmen
   foreach (PropertyItem pi in source.PropertyItems)
   {
      scaledBitmap.SetPropertyItem(pi);
   }

   // Graphics-Objekt für das Bitmap erzeugen und den
   // Interpolier-Modus einstellen
   using (Graphics g = Graphics.FromImage(scaledBitmap))
   {
      // Interpolation bestimmen
      g.InterpolationMode = interpolationMode;

      // Bild von der Quelle auf das Ziel übertragen und dabei skalieren
      g.DrawImage(source, new Rectangle(0, 0, width, height),
         new Rectangle(0, 0, source.Width, source.Height),
         GraphicsUnit.Pixel);
   }

   return scaledBitmap;
}
```

Listing 649: Methode zum Skalieren eines Bilds

Hinweis

In vielen Beispielen zum Skalieren von Bildern ist zu sehen, dass neben dem Interpolier-Modus auch das Anti-Aliasing über die Eigenschaften `SmoothingMode` und `PixelOffsetMode` eingestellt wird. Logisch ist das nicht, denn Anti-Aliasing bezieht sich auf das Zeichnen von schrägen Linien und Kurven und nicht auf das Übertragen von einzelnen Pixeln, wie es bei Bildern der Fall ist. Natürlich habe ich das Ganze auch ausprobiert, um meine Vermutung zu bestätigen und: Die Qualität der resultierenden Bilder ändert sich nach meinen Erkenntnissen nicht, wenn `SmoothingMode` und `PixelOffsetMode` auf eine hohe Qualität eingestellt werden. Deswegen habe ich in der Methode `ScaleBitmap` auch auf diese Einstellung verzichtet.

Eine zweite Variante der `ScaleBitmap`-Methode erwartet statt der neuen Breite und Höhe einen Skalierungsfaktor (und ruft die erste Variante auf):

```
public static Bitmap ScaleBitmap(Bitmap source, double scaleFactor,
   InterpolationMode interpolationMode)
{
   // Neue Breite und Höhe berechnen und damit die erste Variante aufrufen
   int width = (int)(source.Width * scaleFactor);
   int height = (int)(source.Height * scaleFactor);
   return ScaleBitmap(source, width, height, interpolationMode);
}
```

Listing 650: Zweite Variante zum Skalieren eines Bilds

Hinweis

Bei der Anwendung der Skalierungsmethoden sollten Sie ein wenig mit der Interpolation experimentieren. Bei meinen Versuchen hat beim Herunterskalieren eines hochqualitativen JPEG-Photos die Interpolation `High` das beste Ergebnis erbracht. Das Skalieren über ein neues `Bitmap`-Objekt, das ich am Anfang dieses Rezepts erläutert habe, führte hingegen zu einem deutlich schlechteren Ergebnis (was aber beim Skalieren von GIF- oder Bitmap-Bildern nicht unbedingt der Fall war).

Hinweis

Besonders wenn Sie mit JPEG-Dateien experimentieren, die komprimiert gespeichert sind, sollten Sie darauf achten, dass Sie beim Speichern des skalierten Bilds das korrekte Format angeben:

```
scaledBitmap.Save(destFilename, sourceBitmap.RawFormat);
```

Geben Sie das Format nicht an, wird das Bild ansonsten im PNG-Format gespeichert, auch wenn Sie als Dateiendung *.jpg* angeben. Die Größe der Datei ist dann natürlich viel zu groß im Vergleich zum komprimierten JPEG-Format.

328 Thumbnails aus Bildern erzeugen

Thumbnails sind verkleinerte Darstellungen eines Bilds in einer meist festgelegten Größe. Thumbnails werden vorwiegend auf Webseiten dargestellt, sodass der Anwender auf das

Thumbnail klicken und damit das eigentliche Bild öffnen kann. Aber die Erzeugung von Thumbnails ist natürlich auch für C#-Windowsanwendungen interessant.

WPF

In WPF-Anwendungen können Sie Thumbnails über das Skalieren erzeugen, das ich in Rezept 327 beschreiben habe. Falls Sie auf die `Thumbnail`-Eigenschaft der `BitmapDecoder`-Klasse gestoßen sind: Diese ist vorgesehen für Bildformate wie JPEG und TIFF, die Thumbnails neben den Daten des eigentlichen Bilds verwalten können (aber nicht müssen).

Das folgende Beispiel liest eine Bilddatei ein und skaliert diese auf die Größe eines `Image`-Steuerelements (was dieses über die `Stretch`-Eigenschaft natürlich auch selber kann ...). Beim Skalieren wird der Faktor so ausgerechnet, dass die Proportionen des Bilds erhalten bleiben. Das Programm läuft in einer WPF-Anwendung mit den üblichen Referenzen und einem `Image`-Steuerelement, das *image* genannt wird.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Tabou Rocket Ltd 105.jpg");
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.Default);
BitmapSource bitmapSource = decoder1.Frames[0];

// Thumbnail über das Skalieren des Bilds erzeugen
double scaleX = this.image.Width / (double)bitmapSource.Width;
double scaleY = this.image.Height / (double)bitmapSource.Height;
double scaleFactor = Math.Min(scaleX, scaleY);
ScaleTransform scaleTransform = new ScaleTransform(scaleFactor, scaleFactor);
TransformedBitmap transformedBitmap = new TransformedBitmap(bitmapSource,
   scaleTransform);
this.image.Source = transformedBitmap;
```

Listing 651: Skalieren eines Bilds auf die Größe eines Image-Steuerelements

Windows.Forms

In `Windows.Forms`-Anwendungen können Sie Thumbnails über die Methode `GetThumbnailImage` eines `Bitmap`-Objekts erzeugen. Am ersten Argument übergeben Sie die neue Breite, am zweiten die neue Höhe. Das dritte Argument, das einen Delegaten erwartet, wird zurzeit in GDI+ nicht verwendet. Entgegen der Dokumentation können Sie hier einfach `null` übergeben. Am vierten Argument, das ebenfalls nicht verwendet wird, übergeben Sie `IntPtr.Zero`.

Das Beispiel in Listing 652 liest eine JPEG-Datei ein und schreibt das Thumbnail dieser Datei in eine `PictureBox`. Das Beispiel benötigt die Referenzierung der Assembly *System.Drawing.dll* und den Import der Namensräume `System` und `System.Drawing`.

```
string fileName = @"C:\\Bilder\\Les Crosets1.jpg";
Bitmap bitmap = new Bitmap(fileName);
```

Listing 652: Erzeugen eines Thumbnails

```
this.pictureBox1.Image = bitmap.GetThumbnailImage(this.pictureBox1.Width,
   this.pictureBox1.Height, null, IntPtr.Zero);
```

Listing 652: Erzeugen eines Thumbnails (Forts.)

Die Qualität des Thumbnails ist recht gut, kann aber noch verbessert werden. Dazu können Sie einfach die `ScaleBitmap`-Methode aus dem Rezept 327 verwenden:

```
string fileName = @"C:\\Bilder\\Les Crosets1.jpg";
Bitmap bitmap = new Bitmap(fileName);
this.pictureBox2.Image = ScaleBitmap(bitmap, this.pictureBox2.Width,
   this.pictureBox2.Height, InterpolationMode.HighQualityBicubic,
   PixelOffsetMode.HighQuality, SmoothingMode.HighQuality);
```

Listing 653: Erzeugen eines Thumbnails in einer hohen Qualität

Abbildung 173 zeigt eine Windowsanwendung, bei der zwei hochwertige JPEG-Dateien einmal über `GetThumbnailImage` (oben) und dann über `ScaleBitmap` (unten) als Thumbnails dargestellt werden. Die bessere Qualität der unteren Thumbnails können Sie allerdings wahrscheinlich im Buch nicht erkennen.

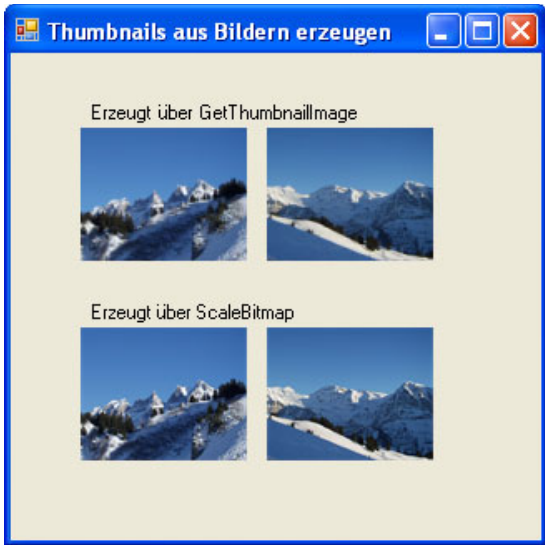


Abbildung 173: Thumbnails von zwei JPEG-Dateien

329 Bilder konvertieren

Das Konvertieren von Bildern ist eigentlich viel zu einfach, als dass dazu ein eigenes Rezept notwendig wäre. Falls Sie aber nicht wissen, wie es geht, zeige ich es hier.

Beim Speichern können Sie auch die Qualität des Bilds beeinflussen, sofern das Bildformat eine Qualitätsangabe unterstützt. Vergleichen Sie dazu das Rezept 330. Beim Konvertieren

müssen Sie darauf achten, dass die Dateigröße entsprechend dem gewählten Format stark variieren kann. Eine Bitmap-Datei ist z. B. wesentlich größer als eine qualitativ gleichwertige JPEG-Datei. Aber das ist ja schon fast Allgemeinwissen ☺.

WPF

In einer WPF-Anwendung lesen Sie das Bild über eine `BitmapDecoder`-Instanz ein (die das Originalformat automatisch erkennt) und speichern Sie dieses über einen der verfügbaren Bild-Encoder ab. Zurzeit stehen die folgenden (aus dem Namensraum `System.Windows.Media.Imaging`) zur Verfügung, deren Format sich größtenteils aus dem Namen ablesen lässt: `BmpBitmapEncoder`, `GifBitmapEncoder`, `JpegBitmapEncoder`, `PngBitmapEncoder`, `TiffBitmapEncoder` und `WmpBitmapEncoder` (Windows-Media-Photo-Format).

Falls Sie auf Ihrem System Codecs für spezielle Formate installiert haben, können Sie deren GUID auch dem Konstruktor der `BitmapEncoder`-Klasse übergeben, um Bilder in einem der nicht direkt unterstützten Formate zu speichern.

Das folgende Beispiel zeigt, wie Sie eine Bilddatei einlesen und im PNG-Format speichern. Das Beispiel benötigt den Import der Namensräume `System`, `System.IO` und `System.Windows.Media.Imaging`.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Karpathos.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.Default);
BitmapSource bitmapSource = decoder.Frames[0];

// Bild im PNG-Format abspeichern
PngBitmapEncoder encoder = new PngBitmapEncoder();
encoder.Frames.Add(BitmapFrame.Create(bitmapSource));
string destFilename = Path.ChangeExtension(imageFileName, ".png");
using (FileStream fileStream = new FileStream(destFilename, FileMode.Create))
{
   encoder.Save(fileStream);
}
```

Listing 654: Einlesen eines JPEG-Bilds und Speichern des Bilds im PNG-Format

Windows.Forms

In einer `Windows.Forms`-Anwendung lesen Sie das Bild in ein `Bitmap`-Objekt ein und speichern dieses über die `Save`-Methode, indem Sie am ersten Argument den Dateinamen und am zweiten Argument das neue Format über einen Wert der `ImageFormat`-Aufzählung angeben.

Das Beispiel in Listing 655 erfordert, dass Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
// JPEG-Datei in Bitmap einlesen
string imageFileName = "C:\\Bilder\\Karpathos.jpg";
Bitmap bitmap = new Bitmap(imageFileName);

// Als PNG-Bild abspeichern
string destFilename = Path.ChangeExtension(imageFileName, ".png");
bitmap.Save(destFilename, ImageFormat.Png);
```

Listing 655: Konvertieren eines JPEG-Bilds in ein PNG-Bild

330 (JPEG-)Bilder mit definierter Qualität speichern

Einige Bildformate wie GIF und JPEG erlauben spezifische Einstellungen, die die Qualität des Bilds beeinflussen. Bei JPEG-Bildern wird die Qualität in Prozent angegeben, GIF ermöglicht Einstellungen wie den Dither-Modus und den Lossy (Verlust).

WPF

Bei den von `BitmapEncoder` abgeleiteten Klassen für die verschiedenen per Default unterstützten Bildformate bieten die Klassen `JpegBitmapEncoder`, `PngBitmapEncoder`, `TiffBitmapEncoder` und `WmpBitmapEncoder` Eigenschaften zur Einstellung der Qualität.

Bei der `JpegBitmapEncoder`-Klasse bestimmt die Eigenschaft `QualityLevel`, die mit einem Wert zwischen 0 und 100 die Qualität in Prozent bestimmt. Die Defaulteinstellung ist 75 (was auch für die Praxis ein sehr guter Kompromiss zwischen Bildqualität und Dateigröße ist).

Die `PngBitmapEncoder`-Klasse ermöglicht, über die Eigenschaft `Interlaced` zu bestimmen, ob das Bild schicht- oder zeilenweise aufgebaut wird. Das hat zwar keinen Einfluss auf die End-Qualität des Bilds, ein Bild im Modus `PngInterlaceOption.On` (schichtweiser Aufbau) wird in einer Webanwendung aber schneller angezeigt, weil die einzelnen Schichten schrittweise von einer relativ geringen zur vollständigen Qualität führen. Die Defaulteinstellung ist `PngInterlaceOption.Default`, was bewirkt, dass die `JpegBitmapEncoder`-Klasse bestimmt, ob Interlacing verwendet wird oder nicht.

Bei der `TiffBitmapEncoder`-Klasse bestimmt die Eigenschaft `Compression` die Einstellung der zu verwendenden TIFF-Komprimierung. Die LZW-Komprimierung (`TiffCompressOption.Lzw`) ist für Farbbilder laut Informationen aus dem Internet die beste und verlustfrei. Die Komprimierungen `Ccitt3`, `Ccitt4` und `Rle` können nur auf Schwarzweiß-Bildern angewendet werden. Andere mögliche Komprimierungen sind `None` (keine) und `Zip`. Die Defaulteinstellung ist `TiffCompressOption.Default`, was bewirkt, dass die `TiffBitmapEncoder`-Klasse versucht, die bestmögliche Komprimierung zu ermitteln.

Die `WmpBitmapEncoder`-Klasse besitzt mehrere Eigenschaften zur Definition der Qualität:

- `AlphaQualityLevel`: Legt mit einem Wert zwischen 0 und 255 die Qualität des »planaren Alphakanals« für Bilder, die einen solchen besitzen, fest. Höhere Werte führen zu einer geringeren Bildqualität.
- `ImageQualityLevel`: Bestimmt die Bildqualität mit einem Wert im Bereich von 0 (schlechte Qualität) bis 1 (verlustfreie Qualität). Der Standardwert ist 0,9.

- `Lossless`: Legt fest, ob die Komprimierung verlustfrei ist. Der Defaultwert ist `false`.
- `QualityLevel`: Bestimmt die Komprimierungs-Qualität für das Hauptbild mit einem Wert zwischen 0 und 255. Höhere Werte führen zu einer geringeren Bildqualität. Der Standardwert ist 1.

Der `GifBitmapEncoder` lässt Qualitäts-Eigenschaften vermissen, obwohl das GIF-Format Qualitäts-Parameter wie den Verlust (Lossy) und Interlacing kennt.

Listing 656 zeigt, wie Sie ein eingelesenes Bild im JPEG-Format mit einer definierten Qualität speichern. Das Beispiel benötigt den Import der Namensräume `System`, `System.IO` und `System.Windows.Media.Imaging`.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Los Roques.png";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.Default);
BitmapSource bitmapSource = decoder.Frames[0];

// Als Jpeg-Bild mit definierter Qualität abspeichern
string destFileName = Path.ChangeExtension(imageFileName, ".jpg");
JpegBitmapEncoder encoder = new JpegBitmapEncoder();
encoder.Frames.Add(BitmapFrame.Create(bitmapSource));
encoder.QualityLevel = 70; // 70% - Default ist 75%
using (FileStream fileStream = new FileStream(destFileName, FileMode.Create))
{
   encoder.Save(fileStream);
}
```

Listing 656: Einlesen eines Bilds und Speichern als JPEG-Bild mit definierter Qualität

Windows.Forms

Beim Speichern von `Bitmap`-Objekten können Sie die Qualität des Bilds nur begrenzt und scheinbar nicht für alle Formate beeinflussen. Leider verlässt .NET an dieser Stelle auch die ansonsten so klare Struktur und wird ein wenig undurchsichtig.

Die grundsätzliche Vorgehensweise erläutert die Methode `SaveAsJpeg` in Listing 657, die ein `Bitmap`-Objekt als JPEG-Bild mit einer gegebenen Qualität speichert.

Dazu ermittelt `SaveAsJpeg` zunächst das für JPEG-Bilder in GDI+ vordefinierte `ImageCodecInfo`-Objekt. Ein solches Objekt verwaltet Informationen darüber, wie die zu erzeugende Datei gespeichert werden muss. Das Ermitteln dieses Objekts ist leider ein wenig aufwändig, weil die Methode alle verfügbaren `ImageCodecInfo`-Objekte durchgehen und deren MIME[1]-Typ verglei-

1. Ein MIME-Typ (Multipurpose Internet Mail Extensions), der ursprünglich für das Versenden von spezifischen Daten in E-Mails vorgesehen war, bezeichnet einen Medientyp und wird hauptsächlich im Internet verwendet, damit der Empfänger weiß, wie die Daten ausgewertet werden müssen. Dazu existieren vordefinierte Bezeichnungen wie z. B. text/html oder image/jpeg.

chen muss. Das `ImageCodecInfo`-Objekt wird lediglich deswegen benötigt, weil die Variante der `Save`-Methode, der ein `EncoderParameters`-Objekt übergeben werden kann, nicht die Angabe des Bildformats über ein `ImageFormat`-Argument erlaubt.

Ein `EncoderParameters`-Objekt, das beim Speichern am letzen Argument übergeben wird, nimmt dann einzelne `EncoderParameter`-Objekte auf, die die Codierung der Dateien bestimmen. Ein `EncoderParameters`-Objekt ist eine Liste solcher Parameter. Bei der Erzeugung geben Sie im Konstruktor die Anzahl der zu definierenden Parameter an. Dann können Sie einzelne `EncoderParameter`-Instanzen an die Liste anhängen, die Sie über die Eigenschaft `Param` erreichen. Bei der Erzeugung eines `EncoderParameter`-Objekts geben Sie am ersten Argument ein `Encoder`-Objekt an, das die Codierungsinformationen speichert. Die `Encoder`-Klasse liefert dazu einige statische Eigenschaften, die die einzelnen Codierungs-Möglichkeiten repräsentieren. Dabei tritt das erste Problem auf: Für die einzelnen Formate unterstützt GDI+ nur eine Auswahl der verfügbaren Codierungen. Beim JPEG-Format sind das z. B. die Codierungen `Encoding.Transformation` (Drehen und Spiegeln des Bilds), `Encoding.Quality` (Einstellen der Qualität), `Encoding.LuminanceTable` und `Encoding.ChrominanceTable`. Leider ist nicht dokumentiert, welches Format welche Codierungsparameter unterstützt. In dem Beispiel zu diesem Rezept und im Repository finden Sie aber eine Hilfsmethode `GetAvailableEncoderNames`, die die Namen der für einen MIME-Typ verfügbaren Codierungsparameter auflistet. Eigenartigerweise werden nach dieser Methode für die Formate *Bitmap*, *GIF* und *PNG* keine Codierungsparameter unterstützt. Zur Vervollständigung finden Sie im Repository auch die Hilfsmethode `GetAvailableMimeTypes`, die die von GDI+ unterstützten MIME-Typen ausliest und zurückgibt.

Am zweiten Argument des `EncoderParameter`-Konstruktors übergeben Sie dann den Wert des Parameters. Bei wenigen Parametern wie `Encoding.Quality` sind Zahlwerte erlaubt. Dabei müssen Sie auch ein wenig ausprobieren. Das zweite Argument des Konstruktors erlaubt u. a. `byte`- und `long`-Werte. Übergeben Sie bei einem `Encoding.Quality`-Parameter einen `byte`-Wert (was eigentlich passen würde, denn die JPEG-Qualität geht nur bis 100 Prozent), resultiert unerklärlicherweise eine Ausnahme mit der Meldung »Ungültiger Parameter verwendet«. Die Konvertierung in einen `long`-Wert bringt dann aber die Lösung dieses Problems.

Andere Parameter wie z. B. `Encoding.Transformation` werden als Wert der `EncoderValue`-Aufzählung übergeben, die alle verfügbaren speziellen Werte auflistet (was die Ermittlung der zu einem Parameter passenden Werte etwas erschwert). Dieser Wert muss in einen `long`-Wert konvertiert werden, da das Argument `value` keinen `EncoderValue`-Typ erlaubt.

Ich habe in `SaveAsJpeg` auf die speziellen Codierungsparameter verzichtet und stelle nur die Qualität ein. Das Drehen und Spiegeln eines `Bitmap`-Objekts, das über den Parameter `Encoding.Transformation` für JPEG-Dateien möglich ist, beschreibt das Rezept 331 unabhängig vom Bildformat.

Mit dem anfangs ermittelten `ImageCodecInfo`-Objekt und den Codierungsparametern wird das Bild dann über die `Save`-Methode des `Bitmap`-Objekts gespeichert.

```
public static void SaveAsJpeg(Bitmap bitmap, string filename, byte quality)
{
   // Das in GDI+ enthaltene ImageCodecInfo-Objekt für JPEG-Bilder
   // ermitteln
   ImageCodecInfo imageCodec = null;
   ImageCodecInfo[] imageCodecs = ImageCodecInfo.GetImageEncoders();
   for (int i = 0; i < imageCodecs.Length; i++)
   {
      if (imageCodecs[i].MimeType=="image/jpeg")
      {
         imageCodec = imageCodecs[i];
         break;
      }
   }

   if (imageCodec == null)
   {
      throw new Exception("Das ImageCodecInfo-Objekt für den " +
         "Mimetyp image/jpeg kann nicht lokalisiert werden");
   }

   // Den wichtigsten der für JPEG unterstützten Encoding-Parameter
   // definieren
   EncoderParameters encoderParams = new EncoderParameters(1);
   encoderParams.Param[0] = new EncoderParameter(
      Encoder.Quality, (long)quality);

   // Bitmap speichern
   bitmap.Save(filename, imageCodec, encoderParams);
}
```

Listing 657: Methode zum Speichern eines Bitmap-Objekts als JPEG-Datei mit definierter Qualität

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System.Drawing` und `System.Drawing.Imaging` importieren.

Die Anwendung ist dann ganz einfach:

```
// PNG-Datei in Bitmap einlesen
string imageFileName = "C:\\Bilder\\Los Roques.png";
Bitmap bitmap = new Bitmap(imageFileName );
// Als Jpeg-Bild mit definierter Qualität abspeichern
string destFileName = Path.ChangeExtension(imageFileName, ".jpg");
SaveAsJpeg(bitmap, destFileName, 70);
```

Listing 658: Anwendung der SaveAsJpeg-Methode

Dabei ist natürlich klar, dass es nichts bringt, wenn Sie Bilder in einer höheren Qualität speichern, als diese im Ursprungszustand besaßen ☺.

331 Bilder drehen, neigen und spiegeln

WPF

In WPF drehen, neigen und spiegeln Sie Bilder (und alles andere, das dargestellt werden kann) über eine Transformation. Die Transformation eines Bilds können Sie wie beim Skalieren von Bildern in Rezept 327 über ein neues `TransformedBitmap`-Objekt ausführen, dem Sie am Konstruktor neben dem zu transformierenden `BitmapSource`-Objekt eine Instanz einer von `Transform` abgeleiteten Klasse übergeben.

Hinweis

Wenn Sie nicht das Bild selbst, sondern lediglich dessen Anzeige drehen oder spiegeln wollen, beachten Sie die in Rezept 344 beschriebene Möglichkeit, GUI-Elemente in WPF zu transformieren.

Zum Drehen verwenden Sie `RotateTransform`. Wenn Sie Bilder transformieren, erlaubt `RotateTransform` aber lediglich die Angabe der Winkel -270, -180, -90, 0, 90, 180 und 270 Grad. Bei allen anderen Winkeln wird eine `InvalidOperationException` geworfen. Wenn Sie allerdings ein GUI-Element rotieren (z. B. ein `Image`-Steuerelement, das ein Bild anzeigt), können Sie jeden Winkel angeben (siehe Beispiel zu diesem Rezept).

Zum Spiegeln verwenden Sie eine `ScaleTransform`-Instanz, der Sie als Skalierungsfaktor für die Breite (`ScaleX`) -1 und für die Höhe (`ScaleY`) 1 übergeben, wenn Sie horizontal (um die X-Achse) spiegeln. Wollen Sie vertikal spiegeln, übergeben Sie für die X-Achse 1 und für die Y-Achse -1.

Beiden Klassen übergeben Sie am Konstruktor oder in Eigenschaften die Transformationswerte. Dazu gehört normalerweise auch die Angabe des Drehpunktes in den Eigenschaften `CenterX` und `CenterY`. Beim Drehen und Spiegeln von Bildern wird der Drehpunkt aber ignoriert, weil der dabei keinen Sinn macht. Beim Drehen und Spiegeln der Anzeige ist der Drehpunkt allerdings wichtig (vgl. Rezept 331).

Wenn Sie mehrere Transformationen ausführen wollen, verwenden Sie dazu ein `TransformationGroup`-Objekt, dem Sie über die `Children`-Eigenschaft einzelne Transformation hinzufügen.

Listing 659 zeigt, wie Sie ein eingelesenes Bild um 90 Grad nach rechts drehen und um die Y-Achse spiegeln.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Hitchhiker.gif");
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];
```

Listing 659: Drehen und Spiegeln eines eingelesenen Bilds in WPF

```
// TransformGroup-Instanz für die Transformationen erzeugen
TransformGroup transformGroup = new TransformGroup();

// RotateTransform-Objekt für das Drehen des Bilds ermitteln
// und der Transformationsgruppe hinzufügen
double rotationAngle = 90;
RotateTransform rotateTransform = new RotateTransform(rotationAngle);
transformGroup.Children.Add(rotateTransform);

// ScaleTransform-Objekt für das Spiegeln des Bilds um die Y-Achse
// ermitteln und der Transformationsgruppe hinzufügen
ScaleTransform scaleTransform = new ScaleTransform(1, -1);
   transformGroup.Children.Add(scaleTransform);
transformGroup.Children.Add(scaleTransform);

// Das eingelesene Bild transformieren
TransformedBitmap transformedBitmap = new TransformedBitmap(
   bitmapSource, transformGroup);
```

Listing 659: Drehen und Spiegeln eines eingelesenen Bilds in WPF (Forts.)

Windows.Forms

Wenn Sie in `Windows.Forms`-Anwendungen Bilder in 90-Grad-Schritten drehen oder an der X- und Y-Achse spiegeln wollen, können Sie dazu die `RotateFlip`-Methode eines `Image`-Objekts verwenden. Dieser Methode übergeben Sie einen Wert der Aufzählung `RotateFlipType`, der die Drehung bzw. das Spiegeln bestimmt. Listing 660 zeigt, wie Sie das in einer `PictureBox` verwaltete Bild um 90 Grad nach rechts drehen. Das Beispiel benötigt die Referenzierung der Assembly *System.Drawing.dll* und die Einbindung der Namensräume `System` und `System.Drawing`.

```
Image image = pictureBox1.Image;
image.RotateFlip(RotateFlipType.Rotate90FlipNone);
pictureBox1.Refresh();
```

Listing 660: Drehen eines Bilds um 90 Grad nach rechts

`RotateFlipType` stellt einige Konstanten zur Verfügung:

- `Rotate90FlipNone`, `Rotate180FlipNone`, `Rotate270FlipNone`: Drehen um 90, 180 bzw. 270 Grad nach rechts ohne zu spiegeln,
- `RotateNoneFlipX`, `RotateNoneFlipY`, `RotateNoneFlipXY`: Spiegeln um die X-, Y- bzw. die X- und die Y-Achse,
- Kombinationen aus einer der möglichen Drehungen und einer Spiegelung in der Form `Rotate90FlipX`, `Rotate90FlipY`, `Rotate90FlipXY`, `Rotate180FlipX` etc.

Die Qualität der Drehung bzw. Spiegelung ist sehr gut und verliert auch nach mehreren Drehungen bzw. Spiegelungen nicht an Wert.

Ewas aufwändiger ist das Drehen eines Bilds um andere Werte als 90, 180 und 270 Grad. Dazu können Sie mehrere Wege gehen. Ein Weg, der auch immer wieder in Newsgroups beschrieben wird, ist die Verwendung eines `Graphics`-Objekts und die Transformation der Zeichenmatrix.

Hinweis

Bevor ich diesen Weg beschreibe, gebe ich einen wichtigen Hinweis: Die .NET-Dokumentation der GDI+-Klassen ist teilweise sehr mager ausgefallen und beschreibt nicht wirklich, was einzelne Klassen und Methoden bewirken. Die eigentliche und häufig aussagekräftigere Dokumentation finden Sie im GDI+-Bereich der Plattform-SDK-Dokumentation. Interessante Artikel zu wichtigen Techniken finden Sie unter GRAPHICS AND MULTIMEDIA SERVICES / GDI+ / USING GDI+ .

Eine Zeichenmatrix definiert die Matrix, auf die sich Positionsangaben beim Zeichnen beziehen. Normalerweise entspricht die Zeichenmatrix der Zeichenfläche. Sie können die Zeichenmatrix aber auch so transformieren, dass Zeichenausgaben verschoben und verdreht erscheinen (und eine andere Skalierung verwenden, aber das ist hier unwichtig).

Die Vorgehensweise ist in der Methode `RotateImage` in Listing 661 implementiert. Bevor Sie sich diese Methode näher anschauen, muss ich zugeben, dass es eine bessere Lösung gibt. Das Problem meiner Lösung ist, dass die Qualität des Bilds unter der Verdrehung ein wenig leidet, was allerdings erst nach einigen nachfolgenden Drehungen sichtbar wird. Die bessere Lösung finden Sie an der Adresse *www.codeproject.com/csharp/rotateimage.asp*.

`RotateImage` berechnet zunächst (mit ein wenig Geometrie) die neue Größe des `Bitmap`-Objekts entsprechend dem angegebenen Winkel, wenn am Argument `resizeBitmap true` übergeben wurde. Damit wird erreicht, dass beim Drehen keine Ecken abgeschnitten werden.

Dann erzeugt `RotateImage` mit den neuen Maßen ein `Bitmap`-Objekt und für dieses ein `Graphics`-Objekt. Das `Graphics`-Objekt wird mit der am Argument `fillColor` übergebenen Farbe gefüllt, damit die überstehenden Ränder des neuen Bitmap an die Hintergrundfarbe des alten angepasst werden können.

Nun kommt der etwas schwierige (und in der .NET-Dokumentation leider sehr spärlich dokumentierte) Teil: Ein `Graphics`-Objekt verwaltet eine Instanz der Klasse `Matrix`. Dieses Objekt, das Sie über die `Transform`-Eigenschaft erreichen, kann Informationen über auszuführende Transformationen beim Zeichnen verwalten. Die vordefinierte `Matrix`-Instanz definiert natürlich keine Transformationen. Sie können aber über verschiedene Methoden Transformationen der Zeichenkoordinaten in X/Y-Richtung, in einem Winkel oder auf eine andere Skalierung erreichen. Das `Graphics`-Objekt bietet dazu direkt die Methoden `TranslateTransform` und `RotateTransform` an. `TranslateTransform` bewirkt eine Transformation der Zeichenmatrix in X/Y-Richtung, `RotateTransform` eine Drehung um einen bestimmten Winkel mit der oberen linken Ecke als Mittelpunkt. Die Zeichenmatrix wird dabei allerdings nicht wirklich verschoben und verdreht: Die Transformationen werden für *jede* nachfolgende Zeichenoperation separat angewendet, indem die Zeichenkoordinaten entsprechend transformiert werden.

Das zweite, leider ebenfalls sehr spärlich dokumentierte Argument der Transformations-Methoden bestimmt, wann welche Transformation angewendet wird. Mit dem Wert `Matrix-`

`Order.Prepend` wird die Transformation vor der vorherigen angewendet, mit dem Wert `MatrixOrder.Append` nach der vorherigen. Dieses Argument ist zur korrekten Anwendung der Transformation sehr wichtig, da es schon ein Unterschied ist, ob die Zeichenmatrix erst verschoben und danach gedreht oder erst gedreht und dann verschoben wird. Sie werden jetzt vielleicht denken: Was soll's – wenn ich erst drehen und dann verschieben will, rufe ich `RotateTransform` einfach vor `TranslateTransform` auf und gebe am zweiten Argument `MatrixOrder.Append` an (das habe ich auf jeden Fall zuerst gedacht). Wenn Sie aber z. B. *mehrfach* mit *derselben* Verschiebung verdrehen wollen, müssen Sie `TranslateTransform` vor `RotateTransform` aufrufen und bei `RotateTransform` am zweiten Argument `MatrixOrder.Prepend` angeben. Dabei sollten Sie bedenken, dass weitere Aufrufe der Transformationsmethoden die Matrix immer weiter verschieben bzw. verdrehen.

Ein wesentlicher Nachteil der `RotateTransform`-Methode für unsere Zwecke ist, dass diese um die obere linke Ecke dreht. Die zur korrekten Ausrichtung des gedrehten Bilds notwendigen Berechnungen waren mir zu komplex. Deshalb bin ich einen anderen Weg gegangen. Sie können das `Matrix`-Objekt nämlich über dessen eigene Methoden wesentlich flexibler direkt einstellen. Die Methode `Translate` bewirkt eine Verschiebung, der Methode `RotateAt` können Sie zum Drehen neben dem Drehwinkel auch die Koordinaten des Drehpunkts übergeben. `RotateImage` initialisiert nun ein neues `Matrix`-Objekt so, dass die Koordinaten beim Zeichnen nach rechts und unten verschoben und um den Mittelpunkt verdreht werden. Das gezeichnete Bild erscheint somit verdreht in der Mitte der Zeichenfläche.

Dabei ist es leider, aus mir unklaren Gründen, nicht möglich, das `Matrix`-Objekt des `Graphics`-Objekts über dessen `Transform`-Eigenschaft direkt anzusprechen (ein Versuch führte zu keinerlei Transformation, aber auch zu keinem Fehler). Sie müssen ein neues `Matrix`-Objekt erzeugen und dieses der `Transform`-Eigenschaft zuweisen, wie es `RotateImage` zeigt.

Vor dem Zeichnen des Quellbilds wird die Ausgabequalität so definiert, dass diese möglichst gut ist. Vergleichen Sie dazu das Rezept 327. Ich habe in `RotateImage` darauf verzichtet, die hierbei verwendeten Werte als Argumente zu übergeben, aber das können Sie ja bei Bedarf nachholen.

Schließlich zeichnet `RotateImage` das Quellbild auf das Ziel-Bitmap, gibt die Ressourcen des `Graphics`-Objekts frei und gibt eine Referenz auf das neue `Bitmap`-Objekt zurück.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Drawing2D` importieren.


```
public static Bitmap RotateImage(Image image, float angle, Color fillColor,
   bool resizeBitmap)
{
   // Neue Breite und Höhe berechnen
   int newHeight, newWidth;
   if (resizeBitmap)
   {
      // Berechnung des Umfassungsrechtecks
      int x = image.Width / 2;
```

Listing 661: Methode zum Drehen eines Bilds (allerdings mit leichten Qualitätsverlusten)

```
      int y = image.Height / 2;
      double cosTheta = Math.Cos(2 * Math.PI * angle / 360);
      double sinTheta = Math.Sin(2 * Math.PI * angle / 360);
      double a = Math.Max(Math.Abs(x * cosTheta + y * sinTheta),
         Math.Abs(x * cosTheta - y * sinTheta));
      double b = Math.Max(Math.Abs(x * sinTheta - y * cosTheta),
         Math.Abs(x * sinTheta + y * cosTheta));
      newWidth = (int)Math.Round(2 * a);
      newHeight = (int)Math.Round(2 * b);
   }
   else
   {
      newHeight = image.Height;
      newWidth = image.Width;
   }

   // Neues Bitmap-Objekt mit den vergrößerten Ausmaßen des alten erzeugen
   Bitmap bitmap = new Bitmap(newWidth, newHeight);

   // Graphics-Objekt für das Bitmap erzeugen und mit der übergebenen
   // Farbe füllen
   Graphics g = Graphics.FromImage(bitmap);
   g.Clear(fillColor);

   // Neue Transformationsmatrix erzeugen
   Matrix matrix = new Matrix();

   // Die Transformation so einstellen, dass die Ausgabe des Bilds
   // in der Mitte der Zeichenfläche erscheint und diese um den
   // Mittelpunkt um den angegebenen Winkel gedreht wird
   int xOffset = (int)((newWidth - image.Width) / 2);
   int yOffset = (int)((newHeight - image.Height) / 2);
   matrix.Translate(xOffset, yOffset);
   Point rotationPoint = new Point(image.Width / 2, image.Height / 2);
   matrix.RotateAt(angle, rotationPoint, MatrixOrder.Prepend);
   g.Transform = matrix;

   // Die Zeichenqualität einstellen
   g.InterpolationMode = InterpolationMode.HighQualityBicubic;
   g.PixelOffsetMode = PixelOffsetMode.HighQuality;
   g.SmoothingMode = SmoothingMode.HighQuality;

   // Das Bild entsprechend der Transformation
   // verschoben und verdreht ausgeben
   g.DrawImage(image, 0, 0, image.Width, image.Height );
```

Listing 661: Methode zum Drehen eines Bilds (allerdings mit leichten Qualitätsverlusten) (Forts.)

```
    // Das Bild zurückgeben
    return bitmap;
}
```

Listing 661: Methode zum Drehen eines Bilds (allerdings mit leichten Qualitätsverlusten) (Forts.)

Abbildung 174 zeigt eine Anwendung, die ein Bild über `RotateImage` um 33 Grad gedreht hat. Um die Drehung und die neue Größe des Bitmap zu verdeutlichen, wurde als Füllfarbe `Color.Silver` übergeben.

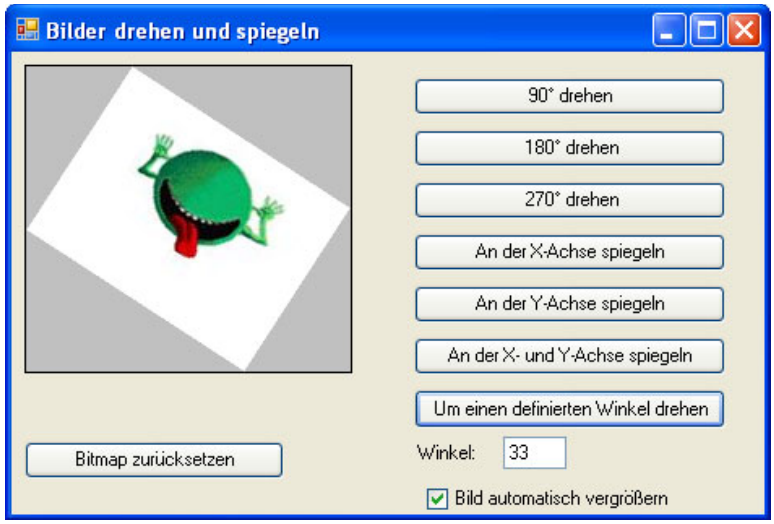


Abbildung 174: Beispielanwendung zum Drehen und Spiegeln eines Bilds mit einem um 33 Grad gedrehten Bild

332 Bildausschnitte auslesen

Wenn Sie beispielsweise ein Verschiebepuzzle programmieren wollen, ist es notwendig, ein Bild in mehrere Teile zu zerschneiden. In WPF-Anwendungen ist diese Aufgabe wesentlich einfacher als in `Windows.Forms`-Anwendungen.

WPF

In WPF verwenden Sie zum Erstellen eines Bildausschnitts eine neue Instanz der `CroppedBitmap`-Klasse (aus dem Namensraum `System.Windows.Media.Imaging`) um einen Bildausschnitt zu erzeugen. Am Konstruktor übergeben Sie das Originalbild und ein `Int32Rect`-Rechteck, das den Bildausschnitt bestimmt. Beachten Sie, dass die Werte in diesem Rechteck scheinbar (leider nicht dokumentiert ...) in Pixeln angegeben sind, und nicht in der ansonsten in WPF üblichen 1/96-Zoll-Einheit (die z. B. die Eigenschaften `Width` und `Height` einer `BitmapSource`-Instanz verwenden). Wenn Sie sich also auf die Breite oder Höhe eines Bilds beziehen, müssen Sie die Eigenschaften `PixelWidth` und `PixelHeight` verwenden.

Das folgende Beispiel schneidet ein 100 * 100 Pixel großes Teil aus der Mitte eines eingelesenen Bilds aus. Es benötigt den Import der Namensräume `System`, `System.Windows.Media.Imaging` und `System.Windows`.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Les Crosets.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];

// Ein 100 * 100 Pixel großes Stück aus der Mitte auslesen
CroppedBitmap croppedBitmap = new CroppedBitmap(bitmapSource,
   new Int32Rect((bitmapSource.PixelWidth - 100) / 2,
   (bitmapSource.PixelHeight - 100) / 2, 100, 100));
```

Listing 662: Einlesen eines Bilds und Erstellen eines Ausschnitts in WPF

Abbildung 175 zeigt eine WPF-Anwendung, die den linken oberen Teil eines Bilds in einzelne Teile zerschnitten und diese in einzelne `Image`-Steuerelemente geschrieben hat.

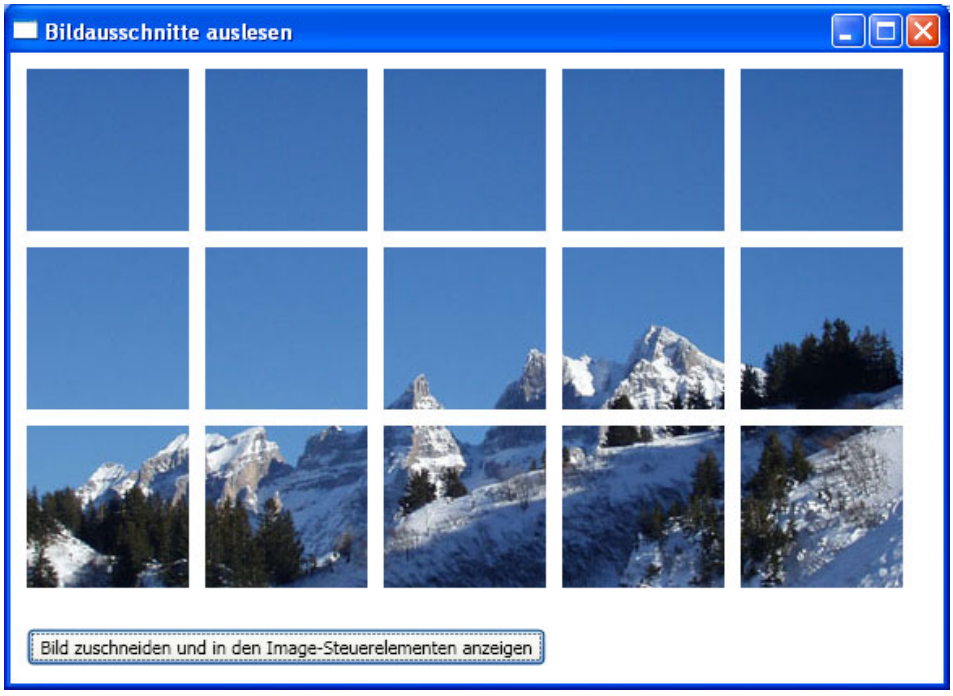


Abbildung 175: Ein Bild wurde in mehrere Teile zerschnitten

Windows.Forms

In `Windows.Forms` können Sie zur Erstellung eines Bildausschnitts den Weg gehen, der in der Methode `CropImage` in Listing 663 implementiert ist. `CropImage` erwartet ein `Image`-Objekt und außerdem die Position und Größe des Ausschnitts.

`CropImage` erzeugt als Erstes ein neues `Bitmap`-Objekt mit den gegebenen Ausmaßen und ein `Graphics`-Objekt für das Zeichnen auf dem Bild. Vor dem Schreiben des Ausschnitts wird die Ausgabequalität so definiert, dass diese möglichst ideal ist. Vergleichen Sie dazu das Rezept 327. Über die `DrawImage`-Methode des `Graphics`-Objekts schreibt `CropImage` schließlich den Ausschnitt aus dem Quellbild in das Ziel-Bitmap.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing`, `System.Drawing.Imaging` und `System.Drawing.Drawing2D` importieren.

```
public static Bitmap CropImage(Image image, int x, int y, int width,
    int height)
{
   // Neues Bitmap-Objekt mit den gegebenen Ausmaßen und dafür ein
   // Graphics-Objekt erzeugen
   Bitmap croppedBitmap = new Bitmap(width, height);
   Graphics g =  Graphics.FromImage(croppedBitmap);

   // Die Zeichenqualität einstellen
   g.InterpolationMode = InterpolationMode.HighQualityBicubic;
   g.PixelOffsetMode = PixelOffsetMode.HighQuality;
   g.SmoothingMode = SmoothingMode.HighQuality;

   // Den Ausschnitt des Quellbilds auf das Ziel-Bild kopieren
   Rectangle destRect = new Rectangle(0, 0, width, height);
   Rectangle sourceRect = new Rectangle(x, y, width, height);
   g.DrawImage(image, destRect, sourceRect, GraphicsUnit.Pixel);
   g.Dispose();

   // Ergebnis zurückgeben
   return croppedBitmap;
}
```

Listing 663: GDI-Methode zum Zuschneiden eines Bilds

333 Farben von Bildern auf andere Farben mappen

Wenn Sie Farben eines Bilds auf andere Farben mappen wollen, können Sie unter GDI ein `ColorMap`-Array verwenden. WPF bietet scheinbar (nach eigener Recherche und nach Postings aus dem Microsoft-WPF-Forum) zurzeit noch keine direkte Möglichkeit, Farben zu transformieren. Sie können unter WPF allerdings die einzelnen Pixel eines Bilds bearbeiten, wie ich es in diesem Rezept zwar einsetze, aber in Rezept 336 erst näher beschreibe. Eine mögliche Art

der Transformation unter WPF ist übrigens das Konvertieren über eine `ColorConvertedBitmap`-Instanz, die den Farbraum des Bilds ändert, und eine `FormatConvertedBitmap`-Instanz, über die Sie das Pixelformat ändern können (z. B. um Graustufen-Bilder zu erzeugen, wie ich es in Rezept 337 zeige).

WPF

Wie gesagt können Sie in WPF Farben mappen, indem Sie die einzelnen Pixel des Bilds durchgehen. Das folgende Beispiel zeigt, wie Sie die Farbe des ersten Pixel eines eingelesenen Bilds auf die Farbe *Navy* mappen. Das Beispiel erfordert den Import der Namensräume `System`, `System.Windows.Media` und `System.Windows.Media.Imaging`.

```
// Bild einlesen
string imageFileName = "C:\\Temp\\Hitchhiker.gif";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];

// Das Bild zunächst in das BGRA-Format konvertieren,
// falls es ein anderes Format aufweist
if (bitmapSource.Format != PixelFormats.Bgra32)
{
   bitmapSource = new FormatConvertedBitmap(bitmapSource,
      PixelFormats.Bgra32, null, 0.0);
}

// Die Pixel einlesen
int stride = bitmapSource.PixelWidth *
   ((bitmapSource.Format.BitsPerPixel + 7) / 8);
byte[] pixelData = new byte[stride * bitmapSource.PixelHeight];
bitmapSource.CopyPixels(pixelData, stride, 0);

// Die Farbe des ersten Pixel ermitteln
byte blue = pixelData[0];
byte green = pixelData[1];
byte red = pixelData[2];

// Die einzelnen Pixel durchgehen und deren Farbe ggf. mappen
for (int i = 0; i < bitmapSource.PixelHeight * stride;
   i += ((bitmapSource.Format.BitsPerPixel + 7) / 8))
{
   if (pixelData[i] == blue &&
      pixelData[i + 1] == green &&
      pixelData[i + 2] == red)
   {
      pixelData[i] = Colors.Navy.B;
```

Listing 664: Mappen von Farben eines Bilds auf andere in WPF

```
            pixelData[i + 1] = Colors.Navy.G;
            pixelData[i + 2] = Colors.Navy.R;
        }
    }

    // Mit den neuen Pixel-Daten ein neues Bild erzeugen
    BitmapSource transformedBitmap = BitmapSource.Create(
        bitmapSource.PixelWidth, bitmapSource.PixelHeight,
        bitmapSource.DpiX, bitmapSource.DpiY, PixelFormats.Bgra32,
        bitmapSource.Palette, pixelData, stride);
```

Listing 664: Mappen von Farben eines Bilds auf andere in WPF (Forts.)

Ein Problem dieser Technik ist, dass sie (besonders bei großen Bildern) nicht besonders schnell ist. Eine mögliche Lösung wäre das Durchgehen der nativen Bilddaten über einen Zeiger, das ich für WPF in Rezept 336 diskutiere und für GDI zeige.

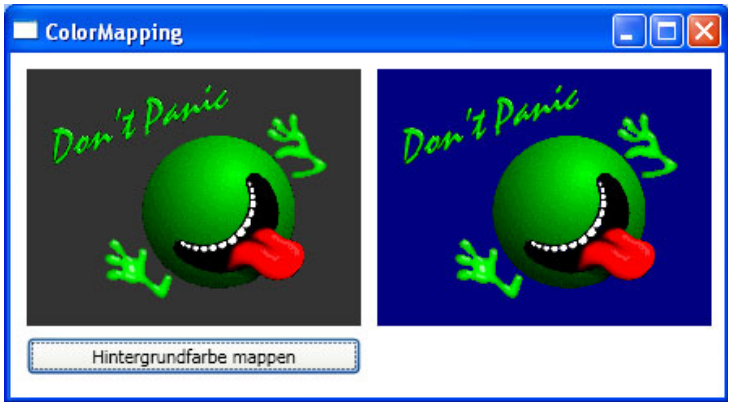


Abbildung 176: Ändern von Farbwerten eines Bilds

Windows.Forms

Zur Umwandlung einzelner Farbwerte eines Bilds in andere können Sie unter GDI ein `System.Drawing.Imaging.ColorMap`-Array verwenden. Die darin verwalteten `ColorMap`-Objekte speichern in ihrer Eigenschaft `OldColor` die alte und in `NewColor` die neue Farbe. Da es sich um ein Array handelt, können Sie beliebig viele Farbumwandlungen definieren.

Dieses `ColorMap`-Array übergeben Sie dann über die `SetRemapTable`-Methode an ein neues `ImageAttributes`-Objekt. Ein solches Objekt verwaltet Informationen über die Veränderung der Bildfarben während der Ausgabe. Neben speziellen Einstellungen wie dem Gammawert können Sie dem `ImageAttributes`-Objekt eben auch ein `ColorMap`-Objekt für Farbumwandlungen übergeben.

Dieses Objekt können Sie dann der `DrawImage`-Methode eines `Graphics`-Objekts am Argument `imageAttrs` übergeben oder beim Erzeugen eines `TextureBrush`-Objekts (für das Zeichnen mit einem Textur-Pinsel) dem Konstruktor übergeben.

Listing 665 zeigt die Vorgehensweise. Das Beispiel liest das in der `PictureBox` *pictureBox1* gespeicherte Bild aus, zeichnet dieses auf ein neues `Bitmap`-Objekt und schreibt das Bild in die `PictureBox` *pictureBox2*. Für die Farbumwandlung wird die Farbe des ersten Pixels ausgelesen und als alte Farbe eines `ColorMap`-Objekts gesetzt. Als neue Farbe wird `Color.Navy` eingesetzt.

Zum Kompilieren dieses Quellcodes benötigen Sie ein Formular mit zwei `PictureBox`-Steuerelementen *pictureBox1* und *pictureBox2*. Außerdem müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
// Bitmap aus der ersten PictureBox auslesen
Bitmap sourceBitmap = (Bitmap)this.pictureBox1.Image;

// Neues Bitmap mit den Ausmaßen des Originals erzeugen
Bitmap destBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

// ColorMap-Array für die Transformation der Farben erzeugen
ColorMap[] map = new ColorMap[1];
map[0] = new ColorMap();
map[0].OldColor = sourceBitmap.GetPixel(0, 0);
map[0].NewColor = Color.Navy;

// Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
// ImageAttributes-Objekt mit dem ColorMap-Objekt übergeben
ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetRemapTable(map);
using (Graphics g = Graphics.FromImage(destBitmap))

{
   g.DrawImage(sourceBitmap, new Rectangle(0, 0, sourceBitmap.Width,
      sourceBitmap.Height), 0, 0, sourceBitmap.Width, sourceBitmap.Height,
      GraphicsUnit.Pixel, imageAttributes);
}

// Neues Bitmap-Objekt in der zweiten PictureBox ablegen
pictureBox2.Image = destBitmap;
```

Listing 665: Ändern einer Hintergrundfarbe eines Bilds

Hinweis

Ein kleines Problem dieser Technik ist, dass immer nur die Farben ersetzt werden, die genau den Farben im `ColorMap`-Array entsprechen. Ein Ersetzen von annähernd gleichen Farben (z. B. allen Rottönen) ist damit nicht bzw. nur mit einer Vielzahl an Definitionen im `ColorMap`-Array möglich.

334 Farbinformationen von Bildern gezielt verändern

Wie schon in Rezept 333 ist das gezielte Verändern von Farben unter WPF scheinbar nicht möglich. Um Farbinformationen gezielt zu verändern, müssen Sie die Pixel eines Bilds bearbeiten. Unter GDI ist eine Farbveränderung allerdings direkt (und sehr schnell) möglich.

WPF

In WPF können Sie einzelne Farben eines Bilds scheinbar nur so verändern, dass Sie die einzelnen Pixel des Bilds bearbeiten. Rezept 333 geht näher darauf ein. Hier zeige ich nur eine Lösung für das Aufhellen eines Bilds um 50%. Das Beispiel erfordert, dass Sie die Namensräume `System`, `System.Windows.Media` und `System.Windows.Media.Imaging` einbinden.

```
// Bild einlesen
string imageFileName = "C:\\Temp\\Hitchhiker.gif";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];
// Das Bild zunächst in das BGRA-Format konvertieren,
// falls es ein anderes Format aufweist
if (bitmapSource.Format != PixelFormats.Bgra32)
{
   bitmapSource = new FormatConvertedBitmap(bitmapSource,
      PixelFormats.Bgra32, null, 0.0);
}

// Die Pixel einlesen
int stride = bitmapSource.PixelWidth *
   ((bitmapSource.Format.BitsPerPixel + 7) / 8);
byte[] pixelData = new byte[stride * bitmapSource.PixelHeight];
bitmapSource.CopyPixels(pixelData, stride, 0);

// Die einzelnen Pixel durchgehen und die Farbe aufhellen
for (int i = 0; i < bitmapSource.PixelHeight * stride;
   i += ((bitmapSource.Format.BitsPerPixel + 7) / 8))
{
   int blue = (int)(pixelData[i] * 1.5);
   if (blue > 255) blue = 255;
   int green = (int)(pixelData[i + 1] * 1.5);
   if (green > 255) green = 255;
   int red = (int)(pixelData[i + 2] * 1.5);
   if (red > 255) red = 255;
   pixelData[i] = (byte)blue;
   pixelData[i + 1] = (byte)green;
   pixelData[i + 2] = (byte)red;
}
```

Listing 666: Gezieltes Verändern der Farben eines Bilds in WPF

```
// Mit den neuen Pixel-Daten ein neues Bild erzeugen
BitmapSource transformedBitmap = BitmapSource.Create(
   bitmapSource.PixelWidth, bitmapSource.PixelHeight,
   bitmapSource.DpiX, bitmapSource.DpiY, PixelFormats.Bgra32,
   bitmapSource.Palette, pixelData, stride);
```

Listing 666: Gezieltes Verändern der Farben eines Bilds in WPF (Forts.)

Abbildung 177 zeigt eine Anwendung, die auf diese Weise das im linken `Image`-Steuerelement dargestellte Bild aufgehellt und in das rechte `Image`-Steuerelement geschrieben hat.



Abbildung 177: Aufhellen eines Bilds

Windows.Forms

Über die Klasse `ColorMatrix` aus dem Namensraum `System.Drawing.Imaging` können Sie die Farb-Informationen eines Bilds sehr flexibel und gezielt verändern. Anders als bei der Verwendung von `ColorMap`-Objekten (Rezept 333) zum Mapping einzelner Farben ermöglicht eine `ColorMatrix`-Instanz die Transformation der ARGB-Werte der einzelnen Pixel des Bilds.

Hinweis

Beim Nachvollziehen dieses Rezepts werden Sie wahrscheinlich feststellen, dass die .NET-Dokumentation der GDI+-Klassen teilweise sehr mager ausgefallen ist. Wenn Sie sich einmal die Dokumentation der Klasse `ColorMatrix` anschauen, werden Sie erkennen, was ich meine: Die Dokumentation sagt zwar etwas über die Bedeutung aus, aber verschweigt die Arbeitsweise der Farbmatrix. Die eigentliche und etwas aussagekräftigere Dokumentation finden Sie im GDI+-Bereich der Plattform-SDK-Dokumentation. Unter GRAPHICS AND MULTIMEDIA SERVICES / GDI+ / USING GDI+ finden Sie viele interessante Artikel, u. a. auch einen zur Verwendung einer Farbmatrix (der aber leider die eigentliche Transformation nicht näher erläutert).

Um die Farben der einzelnen Pixel eines Bilds gezielt zu transformieren, erzeugen Sie eine Instanz der Klasse `ColorMatrix`. Ein solches Objekt verwaltet eine 5*5-Matrix in Form eines verzweigten `float`-Array (`float[][]`). Die Elemente 0 bis 2 der beiden Dimensionen speichern Transformationswerte für den Rot-, Grün- und Blau-Wert, das vierte Element einen Transformationswert für den Alphawert (der die Transparenz bestimmt). Über die fünfte Zeile können

Sie zu den RGB-Farbwerten und zum Alpha-Wert Werte hinzuaddieren. Das Element in der Zelle 4,4 muss laut der Dokumentation immer 1 sein. Die fünfte Spalte wird nicht verwendet und muss laut der Dokumentation (bis auf den Wert in der Zelle 4,4) den Wert 0 speichern.

Eine Farbmatrix, die keinerlei Farbinformationen verändert, sieht z. B. so aus wie in Abbildung 178.

	R	G	B	A	
R	1	0	0	0	0
G	0	1	0	0	0
B	0	0	1	0	0
A	0	0	0	1	0
	0	0	0	0	1

Abbildung 178: Farb-Transformationsmatrix, die keine Farbänderungen bewirkt

Bei einer Farbtransformation werden der Rot-, Grün-, Blau- und der Alpha-Wert jedes Pixels entsprechend der Farbmatrix umgewandelt. Die Transformation ist einfach, aber etwas schwer zu erläutern (weswegen die .NET- und die GDI+-Dokumentation wahrscheinlich auch darauf verzichten). Ich musste recht viel im Internet recherchieren und ein Testprogramm schreiben, um die Transformation zu verstehen. Das Testprogramm ist übrigens ein Bestandteil des Beispiels zu diesem Rezept.

Die Spalten stehen für die Ergebnis-Farbwerte. Die erste Spalte ergibt nach der Transformation den Rotwert, die zweite den Grünwert, die dritte den Blauwert und die vierte den Alphawert. Die Zeilen stehen für die Farbanteile des zu transformierenden Pixels. Der jeweilige Rot-, Grün-, Blau- und Alphawert wird mit dem in der zugehörigen Zeile angegebenen Faktor multipliziert. Die Ergebnisse der einzelnen Multiplikationen werden addiert, wobei die mit 255 multiplizierten Werte der letzten Zeile noch auf das Ergebnis aufaddiert werden (weswegen auch nicht-lineare Transformationen möglich sind). Die aus den einzelnen Spalten resultierenden RGBA-Werte werden dann wieder zu einer Farbe zusammengesetzt und der Pixel damit ausgegeben.

Wahrscheinlich drücken Formeln das Ganze etwas klarer aus. In den Formeln steht Z_0 für die erste Zeile, Z_1 für die zweite etc., S_0 für die erste Spalte, S_1 für die zweite etc.

$$Rot = Rot * (S_0, Z_0) + Grün * (S_0, Z_1) + Blau * (S_0, Z_2) + Alpha * (S_0, Z_3) + (S_0, Z_4) * 255$$

$$Blau = Rot * (S_1, Z_0) + Grün * (S_1, Z_1) + Blau * (S_1, Z_2) + Alpha * (S_1, Z_3) + (S_1, Z_4) * 255$$

$$Grün = Rot * (S_2, Z_0) + Grün * (S_2, Z_1) + Blau * (S_2, Z_2) + Alpha * (S_2, Z_3) + (S_2, Z_4) * 255$$

$$Alpha = Rot * (S_3, Z_0) + Grün * (S_3, Z_1) + Blau * (S_3, Z_2) + Alpha * (S_3, Z_3) + (S_3, Z_4) * 255$$

Das Ergebnis wird bei der Berechnung auf den Byte-Bereich begrenzt, es kommen also nie Werte kleiner 0 oder größer 255 heraus.

Über diese Matrix können Sie nun alle denkbaren Farbtransformationen ausführen. So können Sie den Grünanteil eines Bilds beispielsweise auf 50 Prozent verringern und 80 Prozent des Grünanteils auf den Blauanteil aufaddieren (Abbildung 179).

	R	G	B	A	
R	1	0	0	0	0
G	0	0,5	0,8	0	0
B	0	0	1	0	0
A	0	0	0	1	0
	0	0	0	0	1

Abbildung 179: Farbmatrix zum Verringern des Grünanteils auf 50 Prozent und zum Aufaddieren von 80 Prozent des Grünanteils auf den Blauanteil

Abbildung 180 zeigt eine Matrix, die ein Bild um 20 Prozent aufhellt.

	R	G	B	A	
R	1	0	0	0	0
G	0	1	0	0	0
B	0	0	1	0	0
A	0	0	0	1	0
	0,2	0,2	0,2	0	1

Abbildung 180: Farbmatrix zum Aufhellen eines Bilds um 20 Prozent

Ich verzichte aber an dieser Stelle auf eine weitere Erläuterung der Möglichkeiten und überlasse dies Ihrer Phantasie. Einige Transformationen wie z. B. die Umwandlung eines Bilds in ein Graustufen-Bild setze ich aber in den folgenden Rezepten ein. Hier geht es nun darum, wie Sie die Transformation anwenden. Und das ist recht einfach.

Farbtransformationen können Sie beim Zeichnen über die `DrawImage`-Methode eines `Graphics`-Objekts anwenden, indem Sie wie schon beim `ColorMap`-Array beschrieben am Argument `imageAttr` ein `ImageAttributes`-Objekt übergeben. Eine andere Möglichkeit ist das normale Zeichnen mit einem `TextureBrush`-Pinsel, den Sie bei der Erzeugung mit einem `ImageAttributes`-Objekt initialisieren können (so können Sie z. B. transparente Texturen erzeugen).

Diesem Objekt weisen Sie zuvor über die `SetColorMatrix`-Methode eine `ColorMatrix`-Instanz zu. Und diese müssen Sie lediglich mit einer passenden Matrix initialisieren. Listing 667 zeigt dies am Beispiel eines Bilds, das in einer `PictureBox` gespeichert ist. Das Bild der linken `PictureBox`

wird in diesem Beispiel über eine `ColorMatrix`-Instanz aufgehellt und in die rechte `PictureBox` geschrieben.

Das Beispiel benötigt die Referenzierung der Assembly *System.Drawing.dll* und das Einbinden der Namensräume `System.Drawing` und `System.Drawing.Imaging`.

```
// Bitmap einlesen
string imageFileName = "C:\\Bilder\\Hitchhiker.gif";
Bitmap sourceBitmap = new Bitmap(imageFileName);

// Neues Bitmap mit den Ausmaßen des Originals erzeugen
Bitmap destBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

// ColorMatrix für die Transformation der Farben erzeugen
ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
   new float[] {1, 0, 0, 0, 0},
   new float[] {0, 1, 0, 0, 0},
   new float[] {0, 0, 1, 0, 0},
   new float[] {0, 0, 0, 1, 0},
   new float[] {0.3F, 0.3F, 0.3F, 0, 1}
});

// Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
// ImageAttributes-Objekt mit der ColorMatrix übergeben
ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetColorMatrix(colorMatrix);
using (Graphics g = Graphics.FromImage(destBitmap))
{
   g.DrawImage(sourceBitmap, new Rectangle(0, 0, sourceBitmap.Width,
      sourceBitmap.Height), 0, 0, sourceBitmap.Width, sourceBitmap.Height,
      GraphicsUnit.Pixel, imageAttributes);
}
```

Listing 667: Aufhellen eines Bilds über ein ColorMatrix-Objekt

Das `ColorMatrix`-Objekt muss übrigens nicht unbedingt so initialisiert werden wie in Listing 667. Sie können das Objekt auch über den Default-Konstruktor erzeugen und die Matrix über Eigenschaften definieren, deren Name mit *Matrix* beginnt und mit den Indizes der anzusprechenden Zelle endet. Die Eigenschaft `Matrix01` spricht z. B. die Zelle in der ersten Zeile der zweiten Spalte an. Da der Defaultkonstruktor die Matrix so initialisiert, dass keine Farbänderungen vorgenommen werden, müssen Sie nur die Zellen ändern, die mit einem anderen als dem Defaultwert definiert werden sollen. Die Matrix zum Aufhellen eines Bilds kann z. B. auch so definiert werden:

```
ColorMatrix colorMatrix = new ColorMatrix();
colorMatrix.Matrix40 = 0.3F;
colorMatrix.Matrix41 = 0.3F;
colorMatrix.Matrix42 = 0.3F;
```

335 Ein Negativ eines Bilds erzeugen

Zur Erzeugung eines Bildnegativs können Sie die Technik verwenden, die ich in Rezept 334 beschrieben habe. Um ein Negativ zu erzeugen, müssen Sie die Farbwerte der einzelnen Pixel mit -1 multiplizieren.

WPF

In WPF müssen Sie (leider) wieder die einzelnen Pixel des Bilds von Hand bearbeiten. Die Methode `CreateNegative` in Listing 668 zeigt, wie das geht. Diese Methode erfordert den Import der Namensräume `System.Windows.Media` und `System.Windows.Media.Imaging`.

```
public static BitmapSource CreateNegative(BitmapSource sourceBitmap)
{
   // Das Bild zunächst in das BGRA-Format konvertieren,
   // falls es ein anderes Format aufweist
   if (sourceBitmap.Format != PixelFormats.Bgra32)
   {
      sourceBitmap = new FormatConvertedBitmap(sourceBitmap,
         PixelFormats.Bgra32, null, 0.0);
   }

   // Die Pixel einlesen
   int stride = sourceBitmap.PixelWidth *
      ((sourceBitmap.Format.BitsPerPixel + 7) / 8);
   byte[] pixelData = new byte[stride * sourceBitmap.PixelHeight];
   sourceBitmap.CopyPixels(pixelData, stride, 0);

   // Die einzelnen Pixel durchgehen und die Farbwerte negativieren
   for (int i = 0; i < sourceBitmap.PixelHeight * stride;
      i += ((sourceBitmap.Format.BitsPerPixel + 7) / 8))
   {
      pixelData[i] = (byte)(pixelData[i] * -1);
      pixelData[i + 1] = (byte)(pixelData[i + 1] * -1);
      pixelData[i + 2] = (byte)(pixelData[i + 2] * -1);
   }

   // Mit den neuen Pixel-Daten ein neues Bild erzeugen
   // und zurückgeben
   return BitmapSource.Create(
      sourceBitmap.PixelWidth, sourceBitmap.PixelHeight,
      sourceBitmap.DpiX, sourceBitmap.DpiY, PixelFormats.Bgra32,
      sourceBitmap.Palette, pixelData, stride);
}
```

Listing 668: Methode zur Erzeugung eines Bildnegativs unter WPF

Abbildung 181 zeigt die Beispielanwendung zu diesem Rezept, die aus einem eingelesenen Bild ein Negativ erzeugt hat.

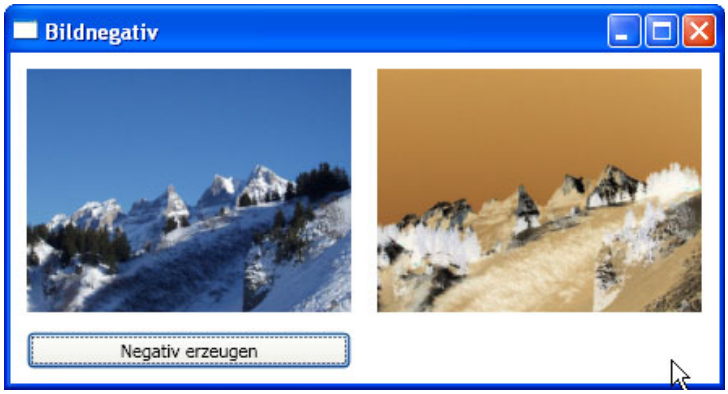


Abbildung 181: Positiv und Negativ eines Landschaftsbilds

Windows.Forms

In einer `Windows.Forms`-Anwendung setzen Sie ein `ColorMatrix`-Objekt (aus dem Namensraum `System.Drawing.Imaging`) ein um ein Negativ zu erzeugen, wie die Methode `CreateNegative` in Listing 669 zeigt.

Zum Kompilieren dieser Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System.Drawing` und `System.Drawing.Imaging` importieren.

```
public static Bitmap CreateNegative(Image image)
{
   // Neues Bitmap mit den Ausmaßen des originalen erzeugen
   Bitmap bitmap = new Bitmap(image.Width, image.Height);

   // ColorMatrix für die Transformation der Farben erzeugen
   ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
      new float[] {-1, 0, 0, 0, 0},
      new float[] {0, -1, 0, 0, 0},
      new float[] {0, 0, -1, 0, 0},
      new float[] {0, 0, 0, 1, 0},
      new float[] {0, 0, 0, 0, 1}
      });

   // Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
   // ImageAttributes-Objekt mit der ColorMatrix übergeben
   ImageAttributes imageAttributes = new ImageAttributes();
   imageAttributes.SetColorMatrix(colorMatrix);
   using (Graphics g = Graphics.FromImage(bitmap))
   {
```

Listing 669: Methode zur Erzeugung eines Bildnegativs unter GDI

```
      g.DrawImage(image, new Rectangle(0, 0, image.Width, image.Height),
         0, 0, image.Width, image.Height, GraphicsUnit.Pixel,
         imageAttributes);
   }

   // Das neue Bild zurückgeben
   return bitmap;
}
```

Listing 669: Methode zur Erzeugung eines Bildnegativs unter GDI (Forts.)

336 Die einzelnen Pixel eines Bilds bearbeiten

In den Fällen, in denen WPF oder GDI keine passende Transformation für die Bearbeitung der Pixel eines Bilds zur Verfügung stellt, können Sie diese auch einzeln bearbeiten. Die Vorgehensweise unterscheidet sich in WPF und `Windows.Forms`.

WPF

In WPF setzen Sie vorwiegend die `CopyPixels`-Methode der `BitmapSource`-Klasse ein um die Pixeldaten in ein Byte-Array zu kopieren. Dieses Array verwenden Sie zur Bearbeitung der einzelnen Pixel. Nach der Bearbeitung erzeugen Sie mit diesen Rohdaten ein neues `BitmapSource`-Objekt.

Hinweis

Alternativ (und wesentlich schneller) können Sie auch die einzelnen Pixel des Bilds über Zeiger in einem unsicheren Bereich bearbeiten. Dies wird von `BitmapSource` aber nicht direkt unterstützt. Ich gehe hier nicht weiter darauf ein. An der Adresse *jmorrill.hjtcentral.com/Home/tabid/428/EntryID/16/Default.aspx* finden Sie einen Artikel, der beschreibt, wie Sie an einen Zeiger auf die nativen Bilddaten herankommen.

Wenn Sie mit `CopyPixels` arbeiten, enthält das erzeugte Array die rohen Daten des Bilds. Das Format ist vom Pixelformat des Bilds abhängig. Da dieses variieren kann, sollten Sie das Bild, das Sie bearbeiten wollen, über eine Transformation in ein für die Bearbeitung sinnvolles Format konvertieren. Ich verwende das `Bgra32`-Format, das pro Pixel vier Byte verwaltet. Das erste Byte definiert den Blauanteil, das zweite den Grünanteil, das dritte den Rotanteil und das vierte den Alpha-Wert (die Transparenz). Das `Bgra32`-Format ist das einzige, das (nur) ein Byte pro Farbe verwendet und einen Alpha-Kanal zur Verfügung stellt.

Zum Kompilieren des folgenden Beispiels müssen Sie die Namensräume `System`, `System.Windows.Media` und `System.Windows.Media.Imaging` importieren.

```
// Bild einlesen
string imageFileName = "C:\\Temp\\Les Crosets.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
```

Listing 670: Einlesen und Konvertieren eines Bilds in das Bgra32-Format

```
   BitmapSource bitmapSource = decoder.Frames[0];

// Das Bild zunächst in das BGRA-Format konvertieren,
// falls es ein anderes Format aufweist
if (bitmapSource.Format != PixelFormats.Bgra32)
{
   bitmapSource = new FormatConvertedBitmap(bitmapSource,
      PixelFormats.Bgra32, null, 0.0);
}
```

Listing 670: Einlesen und Konvertieren eines Bilds in das Bgra32-Format (Forts.)

Zum Auslesen der Pixel (und zum späteren Schreiben) müssen Sie die *Schrittweite* (englisch: Stride) des Bilds in den zu erzeugenden Rohdaten berechnen. Die Schrittweite ist die Anzahl der ganzen Bytes, die eine Zeile im Bild ausmachen. Ein Bild mit einer Auflösung von 32 Bit pro Pixel (bpp) und einer Breite von 100 Pixeln hat demnach eine (Mindest-)Schrittweite von 400 Byte. An der Schrittweite erkennen die Methoden zur Bildverarbeitung und -anzeige, wann eine neue Zeile in den hintereinander gehängten Bytes beginnt.

Die Schrittweite passt nicht immer genau zu den benötigten Bytes pro Zeile. Ein Bild mit einer Breite von elf Pixeln und einer Farbtiefe von vier Bits pro Pixel hat eine Schrittweite von theoretisch 44 Bits = 5,5 Bytes. Da das Byte die kleinste Speichereinheit im (Windows-)System ist, werden (zumindest) sechs Bytes pro Zeile verwendet. Die restlichen vier Bits pro Zeile bleiben einfach leer. In Bilddateien wird aus Performancegründen zusätzlich häufig dafür gesorgt, dass die Anzahl der Bytes pro Zeile ohne Rest durch vier teilbar ist. Im 4-bpp-Bild mit elf Pixeln Breite (6 Bytes) würden also z. B. an jede »Zeile« noch zwei leere Byte angehängt werden.

Wenn Sie selbst Bilder in Byte-Arrays speichern, müssen Sie die Schrittweite bestimmen, in der die Bilder im Array verwaltet werden. Der Grund liegt darin, dass Sie die Schrittweite beim Decodieren wieder angeben müssen, damit der `BitmapDecoder` weiß, wann eine Zeile aufhört. Die Schrittweite sollte natürlich so groß sein, dass eine Bildzeile hineinpasst.

Ich berechne die Schrittweite so, dass unabhängig vom gewählten Pixelformat genügend Platz bleibt:

```
int stride = bitmapSource.PixelWidth *
   ((bitmapSource.Format.BitsPerPixel + 7) / 8);
```


Hinweis

In dem Zusammenhang verstehe ich ehrlich gesagt nicht, warum `CopyPixels` die Schrittweite nicht selbst bestimmt. Die `Create`-Methode der `BitmapDecoder`-Klasse, die Sie später verwenden, um aus dem Byte-Array wieder ein Bild zu erzeugen, könnte ebenfalls ohne die Schrittweite auskommen, weil diese aus der ebenfalls übergebenen Breite und der Farbtiefe berechnet werden kann. Vielleicht lag die Intention darin, dass Sie mit der Schrittweite auch Performance-Optimierungen vornehmen können (durch vier teilbare Bytezahl ...).

Nun können Sie über `CopyPixels` die Pixel einlesen:

```
byte[] pixelData = new byte[stride * bitmapSource.PixelHeight];
bitmapSource.CopyPixels(pixelData, stride, 0);
```

Das letzte Argument gibt einen Offset an, der natürlich 0 ist, wenn Sie das gesamte Bild einlesen wollen.

Dann können Sie das Array durchlaufen und die einzelnen Pixel bearbeiten. Dabei müssen Sie natürlich beachten, dass je nach gewähltem Pixelformat ein Pixel mehr oder weniger Bytes verwendet. In unserem Fall (`Bgra32`) wird ein Pixel in vier Byte definiert und Sie müssen demnach in einer Schleife um jeweils vier Byte weiterspringen. Um zum Pixelformat neutral zu bleiben, berechne ich die Sprungweite mit `(`*bpp* `+ 7) / 8`.

Das Beispiel setzt jedes zweite Pixel auf die Farbe mit den Werten Rot = 0; Grün= 0, Blau = 50 und Alpha = 255:

```
for (int i = 0; i < bitmapSource.PixelHeight * stride;
   i += ((bitmapSource.Format.BitsPerPixel + 7) / 8))
{
   if ((i / 4) % 2 == 0)
   {
      pixelData[i] = 50;      // Blau
      pixelData[i + 1] = 0;   // Grün
      pixelData[i + 2] = 0;   // Rot
      pixelData[i + 3] = 255; // Alpha
   }
}
```

Listing 671: Bearbeiten der einzelnen Pixel

Schließlich erzeugen Sie aus den Rohdaten wieder ein `BitmapSource`-Objekt:

```
BitmapSource transformedBitmap = BitmapSource.Create(
   bitmapSource.PixelWidth, bitmapSource.PixelHeight,
   bitmapSource.DpiX, bitmapSource.DpiY, PixelFormats.Bgra32,
   bitmapSource.Palette, pixelData, stride);
```

Abbildung 182 zeigt das Beispiel zu diesem Rezept, das den hier beschriebenen Code einsetzt.

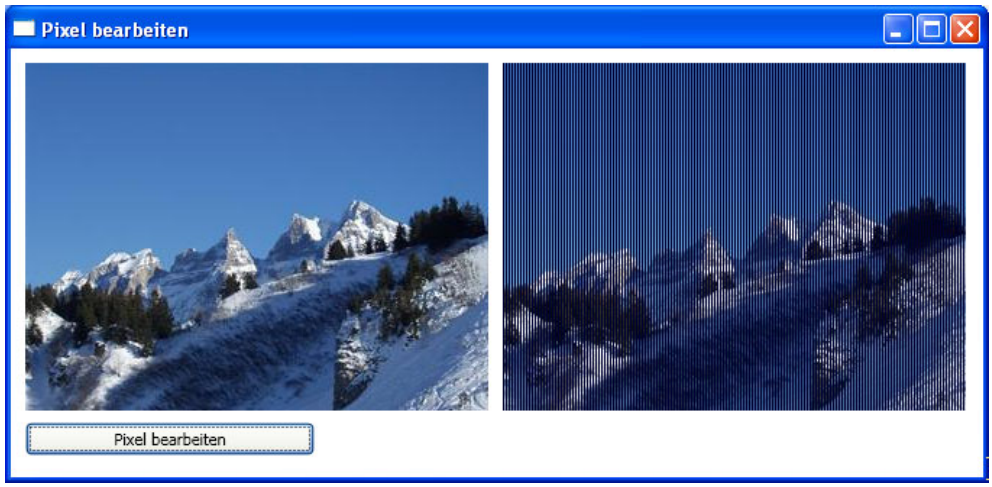


Abbildung 182: Beispiel-Anwendung nach dem Verändern des Bilds

Windows.Forms

In `Windows.Forms`-Anwendungen können Sie die einzelnen Pixel eines `Bitmap`-Objekts über die `GetPixel`-Methode auslesen und (nach einer eventuellen Umrechnung) über `SetPixel` wieder setzen. Listing 672 zeigt, wie Sie damit umgehen. Dieses Beispiel, das die Referenzierung der Assembly *System.Drawing.dll* und den Import der Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` erfordert, definiert jedes zweite Pixel jeder Zeile dunkelblau und erzielt damit einen Streifen-Effekt. Das Beispiel basiert auf einem Windows-Formular mit einer `PictureBox` mit Namen *pictureBox*.

```
// Bitmap referenzieren
Bitmap bitmap = (Bitmap)this.pictureBox.Image;

// Die einzelnen Pixel durchgehen und neu berechnen
Color newPixelColor = Color.FromArgb(0, 0, 50);
for (int x = 0; x < bitmap.Width; x++)
{
   for (int y = 0; y < bitmap.Height; y++)
   {
      if ((x % 2) == 0)
      {
         bitmap.SetPixel(x, y, newPixelColor);
      }
   }
}

// Bild der PictureBox wieder zuweisen
this.pictureBox.Image = bitmap;
```

Listing 672: Direktes Bearbeiten der einzelnen Pixel eines Bilds

Die Verwendung der SetPixel-Methode besitzt einen enormen Nachteil: Sie ist sehr langsam, weil bei jedem Aufruf dieser (und auch der GetPixel-)Methode vor dem Schreiben die Daten des Bitmap-Objekts gesperrt und nach dem Schreiben wieder freigegeben werden. Das Sperren der Daten sorgt dafür, dass die CLR die Bitmap-Daten nicht im Speicher verschieben kann, was sie ansonsten unter Umständen aus Optimierungsgründen machen würde. Das Sperren und Entsperren benötigt einiges an Zeit, weswegen das Schreiben und das Lesen sehr langsam ausgeführt werden.

Eine bessere, weil wesentlich schnellere Lösung ist der direkte Zugriff auf die Bitmap-Daten über einen Zeiger. In diesem Fall müssen Sie die Daten selbst sperren und nach dem Schreiben oder Lesen wieder entsperren, was aber über die LockBits- und die UnlockBits-Methode eines Bitmap-Objekts auf eine einfache Weise möglich ist.

Das folgende Beispiel führt zu demselben Effekt wie das Beispiel mit SetPixel, wird aber wesentlich schneller ausgeführt. Das Beispiel erfordert den Import der Namensräume System, System.Drawing und dass die Einstellung UNSICHEREN CODE ZULASSEN in den Projekteigenschaften eingeschaltet ist.

Zunächst benötigen Sie eine Struktur, in der die Pixeldaten abgelegt werden:

```
private struct PixelData
{
   public byte Blue;
   public byte Green;
   public byte Red;
}
```

Listing 673: Struktur zur Verwaltung von Pixeldaten

Dann referenzieren Sie das Bitmap (in meinem Beispiel, indem dieses aus der PictureBox eines Formulars ausgelesen wird) und sperren dieses über die LockBits-Methode. Dieser Methode müssen Sie am ersten Argument ein Rectangle-Objekt übergeben, das den zu sperrenden Bereich definiert. Da Sie wohl immer das gesamte Bitmap sperren wollen, übergeben Sie hier ein Rechteck, das das gesamte Bild umfasst. Daneben müssen Sie am zweiten Argument den Sperrmodus mit einem Wert der ImageLockMode-Aufzählung definieren. Ich übergebe hier ImageLockMode.ReadWrite, da die einzelnen Pixel sowohl gelesen als auch geschrieben werden sollen. Am letzten Format übergeben Sie noch das Pixel-Format. Ehrlich gesagt habe ich den (undokumentierten) Sinn dieses Arguments nicht verstanden, da LockBits eine Exception wirft, wenn Sie ein Format übergeben, das nicht zum tatsächlichen Format des Bilds passt. Deshalb übergebe ich einfach das Format, das ich aus der PixelFormat-Eigenschaft des Bitmap-Objekts auslese. Das zurückgegebene BitmapData-Objekt benötigen Sie beim Lesen und Schreiben der Daten und für das spätere Entsperren, weswegen Sie dieses in einer Variablen referenzieren müssen. Der Aufruf der LockBits-Methode sollte in einem try-Block erfolgen, damit weiter unten im finally-Block sichergestellt werden kann, dass UnlockBits auch beim Eintritt einer Exception aufgerufen wird.

```
// Bitmap referenzieren
Bitmap bitmap = (Bitmap)this.pictureBox.Image;

// Den Speicher des Bitmaps sperren, damit die CLR diesen nicht
// an einen anderen Platz verlegen kann
BitmapData bitmapData = null;
try
{
   bitmapData = bitmap.LockBits(
      new Rectangle(0, 0, bitmap.Width, bitmap.Height),
      ImageLockMode.ReadWrite, bitmap.PixelFormat);
```

Listing 674: Referenzieren und Sperren des zu verändernden Bitmap

Dann beginnt das Lesen und Schreiben der Bitmap-Daten. Da Sie mit Zeigern arbeiten, muss das Ganze in einem `unsafe`-Block erfolgen. An dieser Stelle verlassen Sie die Sicherheit der CLR und können »wild« im Speicher lesen und schreiben. Um Programmabstürze zu vermeiden sollten Sie hier also sehr vorsichtig programmieren.

Zunächst benötigen Sie einen Zeiger auf die Start-Adresse der Bitmap-Daten. Dazu rufen Sie die `ToPointer`-Methode des `IntPtr`-Objekts auf, das die Eigenschaft `Scan0` verwaltet, und konvertieren dieses in einen Byte-Zeiger.

```
   unsafe
   {
      Byte* pBase = (Byte*)bitmapData.Scan0.ToPointer();
```

Listing 675: Auslesen der Start-Adresse der Bitmap-Daten

Für die Ermittlung der einzelnen Pixel-Daten benötigen Sie die Breite des Bilds in Byte, die sich einfach aus der Pixel-Breite des Bilds multipliziert mit der Größe der `PixelData`-Struktur (drei Byte) berechnet. Dann können Sie die Pixel lesen. Dazu berechnen Sie die Adresse des Pixels, indem Sie die Y-Position mit der Byte-Breite multiplizieren, das Ergebnis der Multiplikation von X mit der Größe der `PixelData`-Struktur addieren und das Ganze schließlich der Start-Adresse hinzuaddieren. Auf diese Weise erhalten Sie die Adresse des jeweiligen Pixels, die Sie noch in einen Zeiger auf `PixelData` konvertieren müssen. Über die Felder `Red`, `Green` und `Blue`, die Sie, da Sie mit einem Zeiger arbeiten, über den Zeiger-Operator (`->`) ansprechen müssen, können Sie den Rot-, Grün-, und Blauwert des Pixels lesen und schreiben:

```
      // Breite des Bilds in Byte berechnen
      int byteWidth = bitmap.Width * sizeof(PixelData);

      // Die einzelnen Pixel durchgehen
      for (int x = 0; x < bitmap.Width; x++)
      {
```

Listing 676: Lesen und Schreiben von einzelnen Pixeln über einen Zeiger

```
            for (int y = 0; y < bitmap.Height; y++)
            {
               if ((x % 2) == 0)
               {
                  // Pixel lesen
                  PixelData* pixelData =
                     (PixelData*)(pBase + (y * byteWidth) +
                     (x * sizeof(PixelData)));

                  // Und farblich anpassen
                  pixelData->Red = 0;
                  pixelData->Green = 0;
                  pixelData->Blue = 50;
               }
            }
         }
      }
   }
```

Listing 676: Lesen und Schreiben von einzelnen Pixeln über einen Zeiger (Forts.)

Schließlich müssen Sie das Bild wieder freigeben:

```
finally
{
   // Bits wieder freigeben
   if (bitmapData != null)
   {
      bitmap.UnlockBits(bitmapData);
   }
}

// Bild der PictureBox wieder zuweisen
this.pictureBox.Image = bitmap;
```

Listing 677: Freigabe der gesperrten Bitmap-Daten und Schreiben des veränderten Bilds in die PictureBox

337 Farb-Bilder in Graustufen-Bilder umwandeln

Für die Umwandlung von Bildern in Graustufen-Bitmaps bietet WPF einen mehr oder weniger direkten Support über das Transformieren des Bilds in das `Gray16`-Pixelformat. In GDI finden Sie kein Äquivalent und müssen selbst programmieren.

WPF

In WPF-Anwendungen können Sie einfach das Pixelformat eines Bilds in ein anderes transformieren um ein Graustufenbild zu erzeugen. Für solche bieten sich die Formate `Gray16` und `Gray32Float` an, die 65.536 bzw. 4 Milliarden Grauschattierungen erlauben.

Zum Konvertieren in ein Graustufenbild erzeugen Sie eine neue Instanz der `FormatConvertedBitmap`-Klasse, der Sie am Konstruktor das zu konvertierende Bild, das neue Pixelformat, `null` am Argument `destinationPalette` und 0 am Argument `alphaThreshold` übergeben.

Das folgende Beispiel, das den Import der Namensräume `System`, `System.Windows.Media` und `System.Windows.Media.Imaging` erfordert, liest ein Bild ein und wandelt es auf diese Weise in ein Graustufenbild um.

```
// Bild einlesen
string imageFileName = "C:\\Bilder\\Les Crosets.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.None);
BitmapSource bitmapSource = decoder.Frames[0];

// Das Bild in ein Graustufenbild umwandeln
FormatConvertedBitmap convertedBitmap = new FormatConvertedBitmap(
   bitmapSource, PixelFormats.Gray16, null, 0);
```

Listing 678: Konvertieren eines Bilds in ein Graustufenbild in WPF

In WPF können Sie natürlich (in meinem Fall leider nur theoretisch) auch in XAML dafür sorgen, dass ein Bild als Graustufenbild angezeigt wird:

```
<Image Height="100">
   <Image.Source>
      <FormatConvertedBitmap DestinationFormat="Gray16">
         <FormatConvertedBitmap.Source>
            <BitmapImage
               UriSource="pack://siteOfOrigin:,,,/Les Crosets.jpg" />
         </FormatConvertedBitmap.Source>
      </FormatConvertedBitmap>
   </Image.Source>
</Image>
```

Listing 679: Anzeige eines konvertierten Bilds über die Deklaration in XAML

Achtung

Obwohl das Beispiel funktionieren sollte, hatte ich (natürlich wieder ...) Probleme. Visual Studio 2008 mochte die Deklaration nicht und meldete den (wirklich sehr aussagekräftigen ...) Fehler »Der Wert liegt außerhalb des erwarteten Bereichs«. Versuche ohne `FormatConvertedBitmap` mit demselben Bild waren (teilweise) erfolgreich und einige sehr komische Dinge passierten während meines Testens. Nach dem Löschen der Zeilen, die `FormatConvertedBitmap` betrafen, und dem nachfolgenden wieder Rückgängigmachen, zeigte Visual Studio auf einmal das in Graustufen umgewandelte Bild an. Daneben funktionierte die Angabe der Bild-Ressource in vielen Fällen auf keine der möglichen Arten (als relativ zum Anwendungsordner angegebener Dateiname wie im obigen Beispiel oder als Ressourcen-Angabe), wenn die Anwendung in einem Ordner mit einem langen Pfad lag. Dieses Problem könnte ich ja noch nachvollziehen, wenn das Bild als Datei (relativ zum Anwendungsordner) angegeben ist. Bei einer in die Anwendungs-Assembly integrierten Ressource sollten allerdings keine Probleme auftreten. Sehr eigenartig und nur sehr schwer nachzuvollziehen, besonders, da nach einem Visual-Studio-Neustart das Bild angezeigt wurde (wenn `FormatConvertedBitmap` nicht verwendet wurde). Da scheint noch einiges an Arbeit notwendig zu sein ...

Auf die Abbildung der Beispielanwendung, die ein Farb-Bild auf diese Weise in ein Graustufen-Bild umwandelt, verzichte ich an dieser Stelle. Sie würden im Buch wohl keinen Unterschied erkennen ☺.

Windows.Forms

Im `Windows.Forms` können Sie die Umwandlung eines Bilds in Graustufen relativ einfach selbst programmieren. Die Grundidee einer einfachen Umwandlung in ein Graustufen-Bild ist, einfach den Mittelwert der RGB-Werte jedes Pixels des Bilds zu berechnen und als Rot-, Grün- und Blauanteil zur Ermittlung eines neuen RGB-Werts zu verwenden. Eine bessere Methode, die unter dem Begriff *Luminanz* bekannt ist, nutzt die Tatsache, dass das menschliche Auge für die einzelnen Grundfarben unterschiedlich empfindlich ist. Diese Methode ermittelt den RGB-Grauwert über die folgende Formel:

$$Y = 0{,}3 * Rot + 0{,}59 * Grün + 0{,}11 * Blau;$$

Das entstehende Graustufenbild entspricht im Kontrast wesentlich mehr dem Originalbild als bei der einfachen Mittelwert-Berechnung.

Um die Farben der einzelnen Pixel neu zu berechnen, können Sie jedes einzelne Pixel auslesen, neu berechnen und wieder zurückschreiben, wie ich es im Rezept 336 beschreibe. Diese Methode, die auch als Lösung im Internet gehandelt wird, ist jedoch recht langsam (auf jeden Fall, wenn Sie dazu reine .NET-Features verwenden). Vergleichen Sie dazu den Artikel »Unsafe image processing« von Eric Gunnerson an der Adresse *blogs.msdn.com/ericgu/archive/2007/06/20/lost-column-2-unsafe-image-processing.aspx*. Eric Gunnerson löst das Geschwindigkeitsproblem hier übrigens über einen guten alten Zeiger.

Sie können das Problem aber auch mit .NET-Mitteln lösen ohne die einzelnen Pixel selbst zu transformieren. Die dazu verwendete Technik beschreibt das Rezept 334.

Die Methode `CreateGrayscaledBitmap` in Listing 680 nutzt diese Technik. Sie erzeugt zunächst ein neues `Bitmap`-Objekt mit den Ausmaßen des übergebenen Bilds und initialisiert ein neues `ColorMatrix`-Objekt mit der Matrix für eine Graustufenumwandlung nach der Luminaz-Methode. Dieses Objekt übergibt `CreateGrayscaledBitmap` dann an ein neues `ImageAttributes`-Objekt. Das übergebene Bild wird schließlich über ein `Graphics`-Objekt in das erzeugte Bitmap geschrieben, wobei am letzten Argument das `ImageAttributes`-Objekt übergeben wird. Beim Übertragen der Pixel wird damit automatisch die im `ColorMatrix`-Objekt definierte Transformation angewendet.

Zum Kompilieren der `CreateGrayscaledBitmap`-Methode müssen Sie die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System`, `System.Drawing` und `System.Drawing.Imaging` importieren.

```
public static Bitmap CreateGrayscaledBitmap(Image image)
{
   // Neues Bitmap-Objekt mit den Ausmaßen der Quelle erzeugen
   Bitmap bitmap = new Bitmap(image.Width, image.Height);

   // ColorMatrix für die Transformation der Farben erzeugen
   ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
      new float[] {0.3F, 0.3F, 0.3F, 0, 0},
      new float[] {0.59F, 0.59F, 0.59F, 0, 0},
      new float[] {0.11F, 0.11F, 0.11F, 0, 0},
      new float[] {0, 0, 0, 1, 0},
      new float[] {0, 0, 0, 0, 1}
      });

   // Grafik auf dem Bitmap-Objekt ausgeben und dabei ein neues
   // ImageAttributes-Objekt mit der ColorMatrix übergeben
   ImageAttributes imageAttributes = new ImageAttributes();
   imageAttributes.SetColorMatrix(colorMatrix);
   using (Graphics g = Graphics.FromImage(bitmap))
   {
      g.DrawImage(image, new Rectangle(0, 0, image.Width, image.Height),
         0, 0, image.Width, image.Height, GraphicsUnit.Pixel,
         imageAttributes);
   }

   return bitmap;
}
```

Listing 680: Methode zur Konvertierung eines Bilds in ein Graustufen-Bitmap unter GDI

Zeichnen

338 GDI-Probleme vermeiden

In manchen `Windows.Forms`-Anwendungen, die einen intensiven Gebrauch von GDI machen, treten beim Zeichnen der Oberfläche eigenartige Probleme auf. Teilweise werden Teile der Oberfläche nicht mehr gezeichnet, teilweise werden Teile der Oberfläche mit eigenartigen Mustern überzeichnet. Diese Probleme nachzuvollziehen ist sehr schwierig, da sie fast ausschließlich in komplexeren Anwendungen auftreten, die viele selbst gezeichnete Steuerelemente einsetzen.

In einer Anwendung, an deren Entwicklung ich beteiligt bin und die fast ausschließlich selbst gezeichnete Steuerelemente einsetzt, trat das GDI-Problem z. B. auf, nachdem der Benutzer einen Splitter mehrfach verschoben hatte, was jeweils zum Neuzeichnen der auf dem Formular angelegten Steuerelemente führte.

Die Ursache des Problems ist wahrscheinlich, dass bei der Programmierung solcher Anwendungen vergessen wurde, die `Dispose`-Methode der verwendeten GDI-Objekte aufzurufen. `Dispose` gibt die von diesen Objekten verwendeten externen (GDI-)Ressourcen frei. Normalerweise sollte `Dispose` vom Finalisierer der Objekte aufgerufen werden, und dieser vom Garbage Collector. Vermutlich zerstört der Garbage Collector diese meist sehr kleinen Objekte aber nicht immer, da es für den Speicher keinen großen Vorteil bringen würde. Dummerweise sind GDI-Ressourcen aber im System begrenzt. Wenn ein Programm die Ressourcen ausgeschöpft hat, können keine neuen reserviert werden, und schon sind die Probleme da.

Die einzige Lösung dieses Problems ist, entweder nach der Verwendung von (GDI-)Objekten deren `Dispose`-Methode aufzurufen, oder (besser) diese in einem `using`-Block zu verwenden (was die Programmierung in der Regel erheblich vereinfacht):

```
using (Brush brush = new SolidBrush(Color.Red))
{
   using (Pen pen = new Pen(Color.Black))
   {
      g.FillRectangle(brush, 10, 10, 100, 100);
      g.DrawRectangle(pen, 10, 10, 100, 100);
   }
}
```

Listing 681: Ressourcen schonendes Zeichnen in using-Blöcken

Diese Regel gilt für alle Objekte, die Sie beim Zeichnen verwenden und selbst erzeugen, also z. B. für Instanzen der Klassen `Brush`, `Pen` und `Font`. In WPF-Anwendungen besitzen die meisten zum Zeichnen verwendeten Grafikobjekte keine `Dispose`-Methode. Die Objekte, die eine solche besitzen, sollten Sie natürlich auch in einem `using`-Block erzeugen, oder Sie sollten nach der Verwendung `Dispose` aufrufen.

Achtung

Die Regel gilt für GDI allerdings nicht für die Klassen `Brushes` und `Pens`, deren statische Eigenschaften Default-`Brush`- bzw. `Pen`-Objekte zurückgeben (wie z. B. `Brushes.Red`, die einen roten `SolidBrush` zurückgibt). Diese Eigenschaften sind so implementiert, dass sie nur beim ersten Zugriff ein neues Objekt erzeugen und eine Referenz darauf zurückgeben. Dieses Objekt wird in einen Thread-internen Cache geschrieben und beim nächsten Zugriff daraus gelesen. Würden Sie diese Objekte in einer `using`-Anweisung verwenden oder die `Dispose`-Methode explizit aufrufen, würde dies beim nächsten Versuch, mit einem solchen Objekt zu zeichnen, in einer `ArgumentException` mit der (wenig aussagekräftigen) Meldung »Ungültiger Parameter« resultieren.

339 Einstellen der Grafik-Qualität unter GDI

Beim Zeichnen unter GDI können (bzw. müssen) Sie die Qualität der erzeugten Grafik über verschiedene Eigenschaften des verwendeten `Graphics`-Objekts beeinflussen. In WPF-Anwendungen ist dies beim Zeichnen von Linien, Kurven und Formen nicht notwendig, da WPF Vektorgrafiken verwendet, die von Natur aus eine sehr hohe Qualität besitzen. Beim skalierten Zeichnen von Bildern wäre eine Angabe der zu verwendenden Interpolation in WPF aber in einigen Fällen wünschenswert.

Unter GDI stehen Ihnen die folgenden Eigenschaften der `Graphics`-Klasse zur Beeinflussung der Qualität zur Verfügung:

- `SmoothingMode`: Über diese Eigenschaft, die einen Wert der `SmoothingMode`-Aufzählung verwaltet, bestimmen Sie das Anti-Aliasing (Kantenglättung) beim Zeichnen von Linien und Kurven. Mit dem Default-Wert `SmoothingMode.Default` werden Kanten nicht geglättet, was dazu führt, dass schräge Linien und Kurven treppenförmig gezeichnet werden. Anti-Aliasing führt dazu, dass die Bereiche rechts und links neben den »Treppen« leicht schattiert oder mit einer schwächeren (leicht transparenten) Farbe aufgefüllt werden. Damit wirken Linien auf den Betrachter glatter bzw. weicher. Über die Konstanten `HighQuality` und `AntiAlias` erreichen Sie die beste Qualität. Daneben können Sie die Konstanten `HighSpeed` (schnelles Zeichnen, aber schlechte Qualität), `None` (keine Kantenglättung) und `Default` (wie `HighSpeed`) verwenden. Etwas verwirrend ist, dass einige der (sehr schlecht dokumentierten) Konstanten zu demselben Ergebnis führen. Nach meinen Versuchen ergeben `HighQuality` und `AntiAlias` eine identisch gute und `Default`, `None` und `HighSpeed` eine identisch schlechte Qualität. Die `Default`-Einstellung ist wahrscheinlich abhängig von einer systemweiten Einstellung, was aber leider nicht dokumentiert ist.
- `PixelOffsetMode`: Diese ebenfalls leider nur sehr schlecht dokumentierte Eigenschaft spezifiziert eine Anti-Aliasing-Erweiterung. Ein Pixeloffset versucht, die Schatten, die das Anti-Aliasing erzeugt, ein wenig weicher zu gestalten. Dabei werden die einzelnen Pixel ein wenig verschoben. Die möglichen Werte der `PixelOffsetMode`-Aufzählung sind wieder etwas verwirrend: Mit `HighQuality` erreichen Sie die beste Qualität, aber auch die schlechteste Geschwindigkeit. `Half` versetzt die Pixel lediglich um eine halbe Einheit und führt zu einem etwas schnelleren Anti-Aliasing als mit `HighQuality`. `HighSpeed`, `None` und `Default` scheinen identische Ergebnisse (keinen Pixeloffset) zu erzielen.

- `InterpolationMode`: Diese Eigenschaft bestimmt die Interpolation, die angewendet wird, wenn Bilder beim Zeichnen vergrößert oder verkleinert werden. Sie hat keinen Einfluss auf das Zeichnen von grafischen Objekten wie Linien, Rechtecke und Kreise. Über die Konstanten `High` und `HighQualityBicubic` erreichen Sie (laut der Hilfe in Photoshop) die beste Qualität. Die bikubische Interpolation ist die langsamste, aber genaueste Methode, über die die gleichmäßigsten Tonabstufungen erzeugt werden. `HighQualityBilinear` führt zu einer etwas verminderten, aber dennoch sehr guten Qualität. Für andere, leider nicht näher dokumentierte Qualitätsstufen stehen Ihnen zudem die Konstanten `Bicubic`, `Bilinear`, `NearestNeighbor`, `Default` und `Low` zur Verfügung. Ob eine hohe Interpolations-Qualität zu einem besseren Ergebnis führt als eine niedrige hängt im Wesentlichen von dem Bild ab, das skaliert werden soll. Bei einigen Bildern ist der optische Eindruck einer qualitativ schlechten Interpolation besser als der einer qualitativ hohen.

- `CompositingMode`: Diese Eigenschaft bestimmt, wie Farben beim Zeichnen mit den Farben des Hintergrundes vermischt werden. Die Defaulteinstellung `SourceOver` legt fest, dass die Farbe eines gezeichneten Objekts mit der Hintergrundfarbe vermischt wird, wobei die Mischung von der Transparenz (also dem Alpha-Wert) der Farbe abhängt. `SourceCopy` legt fest, dass die Farbe die Hintergrundfarbe überschreibt, Transparenz also nicht angewendet wird. Wenn Sie die `CompositingMode`-Eigenschaft auf `CompositingMode.SourceCopy` einstellen, hat dies auch Auswirkungen auf das Anti-Aliasing, das mit Transparenz arbeitet: Die eigentlich geglätteten Kanten sehen dann sehr unschön aus, weil alle Schattierungen durch die solide Farbe des gezeichneten Randes ersetzt werden.

- `CompositingQuality`: Diese Eigenschaft bestimmt die Qualität bei der Kombination von Pixeln beim Zeichnen. Soweit ich weiß, werden Pixel lediglich dann miteinander kombiniert, wenn mehr oder weniger transparent gezeichnet wird. Über `CompositingQuality` können Sie also die Qualität der Transparenz beeinflussen. Bei einer hohen Qualität ist der Kontrast der transparent gezeichneten Objekte wesentlich höher als bei einer niedrigen Qualität. Natürlich reduziert eine hohe Qualität die Ausführungsgeschwindigkeit. Die `CompositingQuality`-Auflistung besitzt die folgenden (sehr schlecht dokumentierten) Konstanten: `AssumeLinear` (gute Qualität über lineare Werte), `Default` (Standard-Qualität), `GammaCorrected` (gute Qualität über eine Gamma-Korrektur), `HighQuality` (hohe Qualität) und `HighSpeed` (niedrige Qualität, nach meinen Erkenntnissen identisch mit `Default`). Leider beeinflusst diese Eigenschaft auch das normale Zeichnen (in der Regel negativ), wenn Anti-Aliasing angewendet wird.

Einige der hier aufgezählten Aufzählungen besitzen noch die weitere Konstante `Invalid`, die einen ungültigen Modus bezeichnet. Diese Konstante können Sie aber nicht zur Einstellung der Qualität verwenden.

Das folgende Beispiel definiert das bestmögliche Anti-Aliasing für das Zeichnen von grafischen Objekten:

```
Graphics g = e.Graphics;
g.SmoothingMode = SmoothingMode.HighQuality;
g.PixelOffsetMode = PixelOffsetMode.HighQuality;
```

Abbildung 183 zeigt jeweils zwei Objekte, die links mit der Default- und rechts mit der optimalen Anti-Aliasing-Qualität gezeichnet wurden.

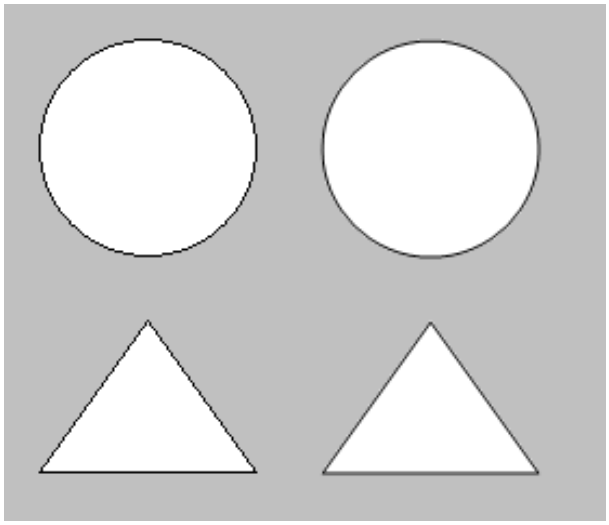


Abbildung 183: Gezeichnete Objekte ohne und mit optimalem Anti-Aliasing

Im nächsten Beispiel wird die Interpolation für das skalierte Zeichnen von Bildern auf eine hohe Qualität eingestellt:

```
Graphics g = e.Graphics;
g.InterpolationMode = InterpolationMode.High;
```

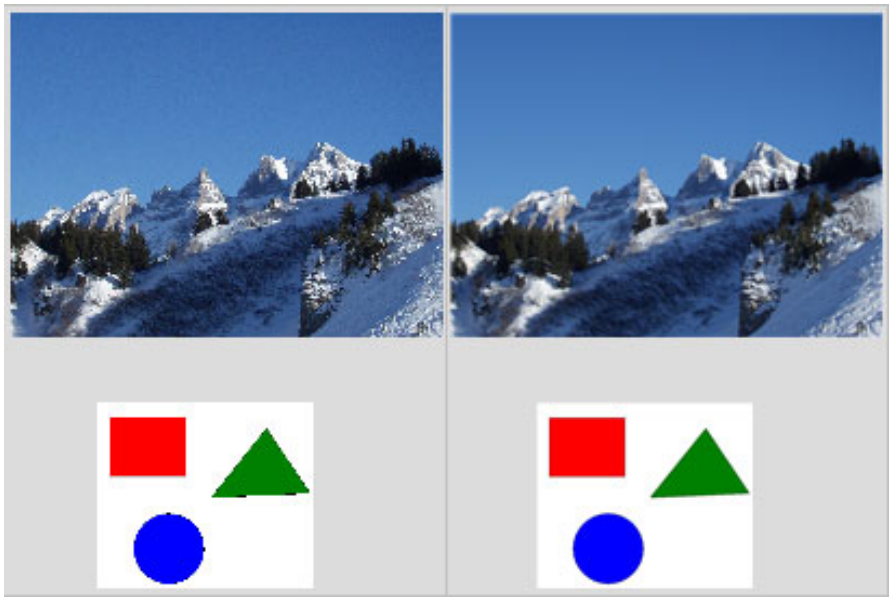


Abbildung 184: Skalierte Bilder (das untere ist ein über Paint erzeugtes Bitmap) ohne und mit Interpolation (InterpolationMode.High)

Das letzte Beispiel stellt die CompositingQuality so ein, dass transparent gezeichnete Objekte mit einem optimalen Kontrast ausgegeben werden:

```
g.CompositingQuality = CompositingQuality.HighQuality;
```

Eine Abbildung spare ich an dieser Stelle ein, da es sehr schwierig ist beim transparenten Zeichnen Qualitätsunterschiede zu erkennen.

Hinweis

In dem Beispiel zu diesem Rezept können Sie die verschiedenen Einstellungen sehr gut ausprobieren, da diese über einzelne ComboBoxen einstellbar sind.

340 Rechtecke mit abgerundeten Ecken zeichnen

WPF

In WPF können Sie komplexe Formen über PathGeometry- oder ein StreamGeometry-Objekt zeichnen. PathGeometry kann in XAML eingesetzt werden, StreamGeometry ist für den Einsatz im Programmcode optimiert und kann auch nur dort eingesetzt werden. Der etwas irreführende Name der StreamGeometry-Klasse kommt übrigens daher, dass diese Klasse intern einen Stream zur Verwaltung ihrer Daten einsetzt (und deswegen weniger Speicher verbraucht als PathGeometry). StreamGeometry erlaubt über Methoden des StreamGeometryContext-Objekts, das Sie über die Open-Methode erhalten, die Definition eines Pfades, der in der Regel den Umriss einer Figur darstellt.

Da es in diesem Kapitel mehr um das programmatische Zeichnen geht (und nicht um die Deklaration von Figuren in XAML), konzentriere ich mich auf die StreamGeometry-Klasse. Zum Zeichnen erzeugen Sie zunächst eine Instanz dieser Klasse und holen dann über die Open-Methode einen StreamGeometryContext. Dieser erlaubt über die Methode BeginFigure den Beginn einer (neuen) Figur. Dabei geben Sie den ersten Punkt der Figur an. Über LineTo können Sie eine Linie hinzufügen, über ArcTo ein Kreissegment. Daneben existieren noch andere Methoden wie BeginFigure, BezierTo, PolyBezierTo und PolyLineTo. Alle Methoden beginnen die Definition des Teilpfades immer da, wo der letzte Teilpfad beendet wurde.

LineTo übergeben Sie am ersten Argument den Zielpunkt, ArcTo übergeben Sie den Zielpunkt, die Größe der Kurve (als Breite und Höhe am Argument size), einen Drehwinkel (rotationAngle), über den Sie die Krümmung einer ovalen Kurve bestimmen können (in unserem Fall nicht erforderlich), eine Information, ob es sich um einen Bogen mit mehr als 180 Grad handelt (isLargeArc) und eine Information darüber, ob der Bogen im Uhrzeigersinn gezeichnet wird (sweepDirection).

An den letzten beiden Argumenten übergeben Sie beiden Methoden eine Information darüber, ob die Linie bzw. Kurve tatsächlich als Line (später, über einen Pen) gezeichnet wird (isStroked) und ob die Linie/Kurve mit angrenzenden Linien/Kurven weich verknüpft wird (isSmoothJoin).

Ist der Pfad fertig definiert, können Sie das StreamGeometry-Objekt über die DrawGeometry-Methode des DrawingContext-Objekts, auf dem Sie die Zeichnung ausgeben wollen, zeichnen. Die Methode DrawRoundedRectangle in Listing 682 nimmt Ihnen diese Arbeit ab.

Diese Methode erwartet an ihren Argumenten das `DrawingContext`-Objekt, auf dem gezeichnet werden soll, die Position und die Ausmaße des zu zeichnenden Rechtecks, den Radius der abgerundeten Ecke, einen `Brush`, der zum Füllen verwendet wird, und einen `Pen`, über den der Rahmen des Rechtecks gezeichnet wird. Für den `Brush` und den `Pen` können Sie auch `null` übergeben, wenn Sie nicht wollen, dass das Rechteck gefüllt bzw. dass der Rand gezeichnet wird.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System.Windows` und `System.Windows.Media` importieren.

```
public static void DrawRoundedRectangle(this DrawingContext dc, int x, int y,
   int width, int height, int cornerRadius, Brush brush, Pen pen)
{
   // StreamGeometry-Instanz erzeugen, die die Geometrie der Form verwaltet,
   // und deren StreamGeometryContext holen
   StreamGeometry geometry = new StreamGeometry();
   using (StreamGeometryContext sgc = geometry.Open())
   {
      // Size-Objekt für die Größe des Viertelkreises der Ecken berechnen
      Size cornerSize = new Size(cornerRadius, cornerRadius);

      // Figur starten
      sgc.BeginFigure(new Point(x + cornerRadius, y), true, true);

      // Linie oben hinzufügen
      sgc.LineTo(new Point(x + width - cornerRadius, y), true, true);

      // Ecke rechts oben hinzufügen
      sgc.ArcTo(new Point(x + width, y + cornerRadius), cornerSize,
         0, false, SweepDirection.Clockwise, true, true);

      // Linie rechts hinzufügen
      sgc.LineTo(new Point(x + width, y + height - cornerRadius), true, true);

      // Ecke rechts unten hinzufügen
      sgc.ArcTo(new Point(x + width - cornerRadius, y + height), cornerSize,
         0, false, SweepDirection.Clockwise, true, true);

      // Linie unten hinzufügen
      sgc.LineTo(new Point(x + cornerRadius, y + height), true, true);

      // Ecke links unten hinzufügen
      sgc.ArcTo(new Point(x, y + height - cornerRadius), cornerSize,
         0, false, SweepDirection.Clockwise, true, true);

      // Linie links hinzufügen
      sgc.LineTo(new Point(x, y + cornerRadius), true, true);
```

Listing 682: Methode zum Zeichnen eines Rechtecks mit abgerundeten Ecken unter WPF

```
      // Ecke links oben hinzufügen
      sgc.ArcTo(new Point(x + cornerRadius, y), cornerSize,
         0, false, SweepDirection.Clockwise, true, true);
   }

   // Das Geometry-Objekt zeichnen
   dc.DrawGeometry(brush, pen, geometry);
}
```

Listing 682: Methode zum Zeichnen eines Rechtecks mit abgerundeten Ecken unter WPF (Forts.)

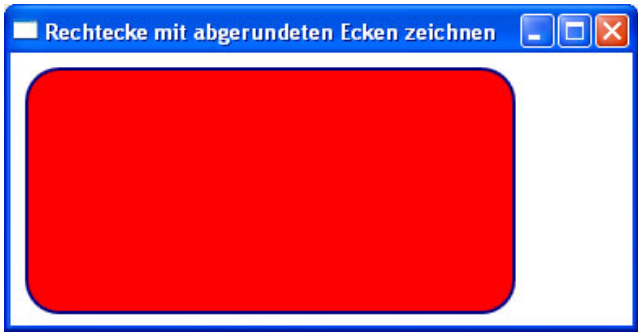


Abbildung 185: Eine Beispiel-Anwendung hat ein Rechteck mit abgerundeten Ecken gezeichnet

GDI

Unter GDI+ zeichnen Sie ein abgerundetes Rechteck ähnlich wie unter WPF. Zur Definition der Figur verwenden Sie ein `GraphicsPath`-Objekt. Ähnlich wie bei WPF fügen Sie über `AddLine` Linien und über `AddArc` Kreisbögen hinzu. Der Unterschied zu WPF ist, dass Sie hier immer auch den Startpunkt der Linie bzw. Kurve angeben und dass Sie die Figur am Ende abschließen müssen.

Die Methode `DrawRoundedRectangle` in Listing 683 zeigt, wie Sie dies programmieren. Sie erwartet am ersten Argument das `Graphics`-Objekt, auf dem gezeichnet werden soll. Die anderen Argumente entsprechen denen der WPF-Variante. `DrawRoundedRectangle` erfordert den Import der Namensräume `System.Drawing` und `System.Drawing.Drawing2D`.

```
public static void DrawRoundedRectangle(this Graphics g,int x, int y,
   int width, int height, int cornerRadius, Brush brush, Pen pen)
{
   // Neuen GraphicsPath erzeugen ...
   GraphicsPath gp = new GraphicsPath();

   // Linie oben hinzufügen
   gp.AddLine(x + cornerRadius, y, x + width - cornerRadius, y);
```

Listing 683: Methode zum Zeichnen eines Rechtecks mit abgerundeten Ecken unter GDI

```
    // Ecke rechts oben hinzufügen
    gp.AddArc(x + width - cornerRadius, y, cornerRadius,
       cornerRadius, 270, 90);

    // Linie rechts hinzufügen
    gp.AddLine(x + width, y + cornerRadius, x + width,
       y + height - cornerRadius);

    // Ecke rechts unten hinzufügen
    gp.AddArc(x + width - cornerRadius, y + height - cornerRadius,
       cornerRadius, cornerRadius, 0, 90);

    // Linie unten hinzufügen
    gp.AddLine(x + width - cornerRadius, y + height,
       x + cornerRadius, y + height);

    // Ecke links unten hinzufügen
    gp.AddArc(x, y + height - cornerRadius, cornerRadius,
       cornerRadius, 90, 90);

    // Linie links hinzufügen
    gp.AddLine(x, y + height - cornerRadius, x, y + cornerRadius);

    // Ecke links oben hinzufügen
    gp.AddArc(x, y, cornerRadius, cornerRadius, 180, 90);

    // Die Figur abschließen
    gp.CloseFigure();

    // Den Pfad mit dem übergebenen Pinsel füllen
    if (brush != null)
    {
       g.FillPath(brush, gp);
    }

    // Die Linien des Pfades zeichnen
    if (pen != null)
    {
       g.DrawPath(pen, gp);
    }
}
```

Listing 683: Methode zum Zeichnen eines Rechtecks mit abgerundeten Ecken unter GDI (Forts.)

Die Methode setzt bewusst nicht die Grafik-Qualität (siehe Rezept 339), damit Sie vor dem Aufruf dieser Methode selbst entscheiden können, wie Sie die Qualität einstellen wollen.

341 Pfeile zeichnen

GDI unterstützt das Zeichnen sehr einfacher Pfeile über das Zeichnen einer Linie über einen `Pen` mit einer Start- oder Endkappe vom Typ `LineCap.ArrowAnchor`. Solche Pfeile sind sehr einfach und nicht besonders schön. Ich verzichte hier auf eine weitere Beschreibung, weil meine Lösung wesentlich eleganter ist (☺) und auch unter WPF funktioniert. Das `Windows.Forms`-Beispiel zu diesem Rezept zeigt aber auch, wie Sie einfache Pfeile zeichnen.

WPF bietet übrigens, soweit ich herausgefunden habe, keine Möglichkeit, einfache Pfeile zu zeichnen. Sie können bei WPF-`Pen`-Objekten zwar auch die Start- und End-Kappe definieren, diese können Sie aber nur auf flach, quadratisch, rund und dreieckig einstellen.

Die Grundidee meiner Lösung war, einen Pfeil zunächst als Figur (in einem Pfad) so zu definieren, dass dieser senkrecht (oder waagerecht) steht, damit die einzelnen Eckpunkte einfacher definiert werden konnten. Die Methode `DrawArrow` (in Listing 684) schreibt die für einen Pfeil notwendigen Punkte (die Eckpunkte und gegebenenfalls zwei Punkte in der Mitte der Pfeilspitze für eine gebogene Spitze) abhängig von den Werten der einzelnen Argumente dazu zunächst in ein `Point`-Array. Um den Pfeil an die durch die Argumente `startPoint` und `endPoint` angegebene Position zu verschieben, wird eine `Matrix` verwendet. Der Rotierwinkel wird dabei bezogen auf den Mittelpunkt des Pfeils und die 12-Uhr-Position des Koordinatensystems berechnet. Neben der Verdrehung (um den Pfeil-Mittelpunkt) ist dann nur noch eine Verschiebung des Pfeils notwendig. Der mit den Punkten erzeugte Pfad wird schließlich mit Hilfe dieser Matrix transformiert.

Ich stelle hier lediglich die WPF-Variante der Methode `DrawArrow` vor. Die GDI-Variante arbeitet ähnlich und ist im Repository, in der Codebook-Klassenbibliothek und im `Windows.Forms`-Beispiel zu diesem Rezept zu finden. Die Argumente der WPF-Version sind die folgenden:

- `dc`: Das `DrawingContext`-Objekt, auf dem gezeichnet werden soll
- `startPoint`: Der Startpunkt des Pfeils
- `endPoint`: Der Endpunkt des Pfeils (Ende der Pfeilspitze)
- `shaftWidth`: Die Breite des Pfeilschafts
- `arrowHeadLength`: Gibt die Länge der Pfeilspitze bis zum linken bzw. rechten Eckpunkt an
- `arrowHeadLengthFromShaft`: Gibt die Länge der Pfeilspitze auf der Y-Linie an, die die Mitte des Pfeilschafts bildet
- `arrowHeadWidth`: Gibt die Breite der Pfeilspitze am Anfang derselben an
- `arrowHeadMiddleWidth`: Gibt die Breite der Pfeilspitze in der Mitte derselben an
- `brush`: Der `Brush`, mit dem der Pfeil gefüllt werden soll. Kann `null` sein.
- `pen`: Der `Pen`, der den Rand des Pfeils definiert. Kann `null` sein.

Ein Wert größer 0 in `arrowHeadMiddleWidth` führt dazu, dass die Seitenlinien der Pfeilspitze als Kurve gezeichnet werden. Bei einem Wert kleiner/gleich 0 werden diese Linien als Gerade gezeichnet.

`DrawArrow` benötigt den Import der Namensräume `System`, `System.Windows` und `System.Windows.Media`.

```
public static void DrawArrow(this DrawingContext dc, Point startPoint,
   Point endPoint, double shaftWidth, double arrowHeadLength,
   double arrowHeadLengthFromShaft, double arrowHeadWidth,
   double arrowHeadMiddleWidth, Brush brush, Pen pen)
{
   // Länge des Pfeils berechnen
   double x1 = Math.Abs(startPoint.X);
   double y1 = Math.Abs(startPoint.Y);
   double x2 = Math.Abs(endPoint.X);
   double y2 = Math.Abs(endPoint.Y);
   double a = x2 - x1;
   double b = y2 - y1;
   double arrowLength = Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 2));

   // Der Pfeil wird virtuell so gezeichnet, dass die Spitze
   // auf dem Punkt (0,0) liegt und der Anfang auf dem Punkt (0, Pfeillänge):
   // Virtuellen Start- und Endpunkt berechnen
   double widthOffset = (shaftWidth / 2);
   Point virtualEndPoint = new Point(0, 0);
   Point virtualStartPoint = new Point(virtualEndPoint.X,
      virtualEndPoint.Y + arrowLength);
   Point virtualCenterPoint = new Point(virtualEndPoint.X,
      virtualEndPoint.Y + (int)(arrowLength / 2F));

   // Den Mittelpunkt des Pfeils ermitteln
   double xDiff = (endPoint.X - startPoint.X) / 2F;
   double yDiff = (endPoint.Y - startPoint.Y) / 2F;
   Point centerPoint = new Point(startPoint.X + xDiff, startPoint.Y + yDiff);

   // Der Pfeil wird beim Zeichnen so transformiert, dass er korrekt gedreht
   // und verschoben gezeichnet wird:

   // Den Rotationswinkel für die notwendige Drehung berechnen
   Point rotationPoint = new Point(endPoint.X - centerPoint.X,
      endPoint.Y - centerPoint.Y);
   int angleOffset;
   // Grundrechnung: Tan(alpha) = b / a => alpha = Atan(b / a)
   if (rotationPoint.X >= 0 && rotationPoint.Y < 0)
   {
      // Erster Quadrant
      a = rotationPoint.X;
      b = rotationPoint.Y * -1;
      angleOffset = 90;
   }
   else if (rotationPoint.X >= 0 && rotationPoint.Y >= 0)
```

Listing 684: Methode zum Zeichnen eines Pfeils unter WPF

```
{
   // Zweiter Quadrant
   a = rotationPoint.Y;
   b = rotationPoint.X;
   angleOffset = 180;
}
else if (rotationPoint.X < 0 && rotationPoint.Y >= 0)
{
   // Dritter Quadrant
   a = rotationPoint.X * -1;
   b = rotationPoint.Y;
   angleOffset = 270;
}
else
{
   // Vierter Quadrant
   b = rotationPoint.X * -1;
   a = rotationPoint.Y * -1;
   angleOffset = 360;
}

// Winkel im Bogenmaß berechnen
double radian = Math.Atan(b / a);

// Winkel in Grad umrechnen
double rotationAngle = angleOffset - (radian * (180 / Math.PI));

// Matrix erzeugen und die Rotation anwenden
Matrix matrix = new Matrix();
matrix.RotateAt(rotationAngle, virtualCenterPoint.X, virtualCenterPoint.Y);

// Verschiebung berechnen und anwenden
double offsetX = centerPoint.X - virtualCenterPoint.X;
double offsetY = centerPoint.Y - virtualCenterPoint.Y;
matrix.Translate(offsetX, offsetY);

// Punkte für den Pfeil zusammenstellen
Point[] arrowPoints;
if (arrowHeadMiddleWidth > 0)
{
   arrowPoints = new Point[10];
   // Linkes unteres Ende des Pfeilschafts
   arrowPoints[0] = new Point(widthOffset * -1, virtualStartPoint.Y);
   // Linkes oberes Ende des Pfeilschafts
   arrowPoints[1] = new Point(arrowPoints[0].X, arrowHeadLength);
```

Listing 684: Methode zum Zeichnen eines Pfeils unter WPF (Forts.)

```
      // Linke untere Ecke der Pfeilspitze
      arrowPoints[2] = new Point((arrowHeadWidth / 2) * -1,
         arrowHeadLengthFromShaft);
      // Mitte der Pfeilspize links (für eine gekrümmte Pfeilspitze)
      arrowPoints[3] = new Point(-(arrowHeadMiddleWidth / 2),
         arrowHeadLengthFromShaft / 2);
      // Pfeilspitze
      arrowPoints[4] = new Point(0, 0);
      // Mitte der Pfeilspitze rechts (für eine gekrümmte Pfeilspitze)
      arrowPoints[5] = new Point(arrowHeadMiddleWidth / 2,
         arrowHeadLengthFromShaft / 2);
      // Rechte untere Ecke der Pfeilspitze
      arrowPoints[6] = new Point(arrowHeadWidth / 2,
         arrowHeadLengthFromShaft);
      // Rechtes oberes Ende des Pfeilschafts
      arrowPoints[7] = new Point(arrowPoints[1].X + shaftWidth,
         arrowPoints[1].Y);
      // Rechtes unteres Ende des Pfeilschafts
      arrowPoints[8] = new Point(arrowPoints[0].X + shaftWidth,
         arrowPoints[0].Y);
   }
   else
   {
      arrowPoints = new Point[7];
      // Linkes unteres Ende des Pfeilschafts
      arrowPoints[0] = new Point(widthOffset * -1, virtualStartPoint.Y);
      // Linkes oberes Ende des Pfeilschafts
      arrowPoints[1] = new Point(arrowPoints[0].X, arrowHeadLength);
      // Linke untere Ecke der Pfeilspitze
      arrowPoints[2] = new Point((arrowHeadWidth / 2) * -1,
         arrowHeadLengthFromShaft);
      // Pfeilspitze
      arrowPoints[3] = new Point(0, 0);
      // Rechte untere Ecke der Pfeilspitze
      arrowPoints[4] = new Point(arrowHeadWidth / 2,
         arrowHeadLengthFromShaft);
      // Rechtes oberes Ende des Pfeilschafts
      arrowPoints[5] = new Point(arrowPoints[1].X + shaftWidth,
         arrowPoints[1].Y);
      // Rechtes unteres Ende des Pfeilschafts
      arrowPoints[6] = new Point(arrowPoints[0].X + shaftWidth,
         arrowPoints[0].Y);
   }

   // StreamGeometry-Instanz erzeugen, die die Geometrie der Form verwaltet,
```

Listing 684: Methode zum Zeichnen eines Pfeils unter WPF (Forts.)

```
   // und deren StreamGeometryContext holen
   StreamGeometry geometry = new StreamGeometry();
   using (StreamGeometryContext sgc = geometry.Open())
   {
      sgc.BeginFigure(arrowPoints[0], true, true);
      if (arrowHeadMiddleWidth > 0)
      {
         sgc.LineTo(arrowPoints[1], true, true);
         sgc.LineTo(arrowPoints[2], true, true);
         sgc.BezierTo(arrowPoints[2], arrowPoints[3], arrowPoints[4],
            true, true);
         sgc.BezierTo(arrowPoints[4], arrowPoints[5], arrowPoints[6],
            true, true);
         sgc.LineTo(arrowPoints[7], true, true);
         sgc.LineTo(arrowPoints[8], true, true);
      }
      else
      {
         // Aus den Punkten einen Pfad mit gerader Pfeilspitze bilden
         sgc.PolyLineTo(arrowPoints, true, true);
      }
   }

   // Die Transformation anwenden
   geometry.Transform = new MatrixTransform(matrix);

   // Das Geometry-Objekt zeichnen
   dc.DrawGeometry(brush, pen, geometry);
}
```

Listing 684: Methode zum Zeichnen eines Pfeils unter WPF (Forts.)

Abbildung 186 zeigt einige mit `DrawArrow` (über Zufallswerte) gezeichnete Pfeile.

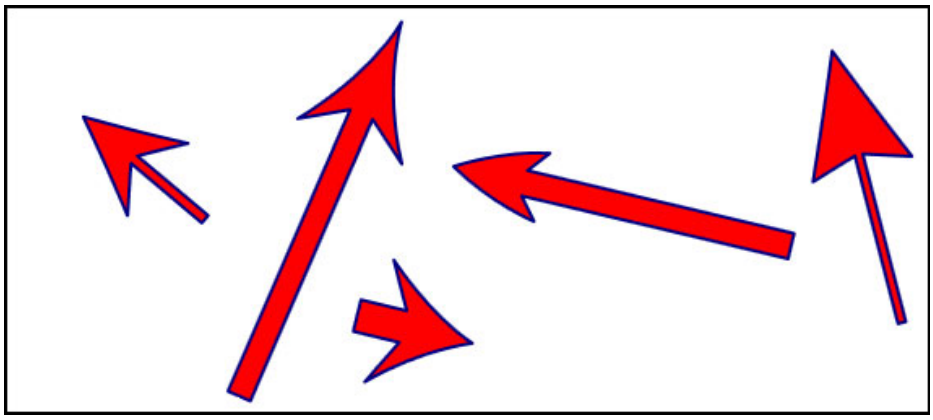


Abbildung 186: Einige mit DrawArrow gezeichnete eigene Pfeile

342 Transparente Bilder und Grafiken erzeugen

Die Transparenz von Grafiken, die Sie ausgeben, können Sie bestimmen, indem Sie den Alphawert der Zeichenfarbe angeben. Mit einem Alphawert von 255 werden Farben undurchsichtig ausgegeben, ein Alphawert von 0 steht für eine vollständige Transparenz. Die Werte dazwischen definieren eine mehr oder weniger starke Transparenz. Eine Farbe mit einem definierten Alphawert können Sie über die `FromArgb`-Methode der `Color`-Struktur (für WPF aus dem Namensraum `System.Windows` und für `Windows.Forms` aus dem Namensraum `System.Drawing`) erzeugen, indem Sie in der Variante mit vier Argumenten am ersten Argument den Alphawert übergeben. Listing 685 zeigt dies am Beispiel eines WPF-Fensters, in dem zunächst ein halb-transparentes Rechteck und ein halb-transparenter Text gezeichnet werden. In Listing 686 finden Sie die entsprechende `Windows.Forms`-Lösung.

WPF

In WPF können Sie zum transparenten Zeichnen von Bildern leider keine Transformation verwenden (siehe Rezept 333). Sie können aber die Pixel des Bilds einzeln bearbeiten, um die Transparenz jedes einzelnen Pixels zu bestimmen (siehe Rezept 336). Listing 685 verwendet diese Technik, um den Hintergrund eines eingelesenen Bilds voll- und die restlichen Pixel halb-transparent zu definieren und das Bild schließlich zu zeichnen.

Das Beispiel läuft in einem WPF-Fenster, das ein `Image`-Steuerelement enthält. Die Quelle des Steuerelements ist auf eine Ressource der Anwendung gelegt, die *Moon.jpg* heißt (und die ein Bild des Mondes ist). Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading`, `System.Windows`, `System.Windows.Media` und `System.Windows.Media.Imaging`.

```
public partial class StartWindow : Window
{
   /* DrawingVisual, auf dem gezeichnet werden kann */
   private DrawingVisual drawingVisual = new DrawingVisual();

   /* Konstruktor. Erzeugt die Steuerelemente und Komponenten und
      die Zeichnung in dem DrawingVisual. */
   public StartWindow()
   {
      InitializeComponent();

      // Den DrawingContext für den DrawingVisual holen
      using (DrawingContext dc = this.drawingVisual.RenderOpen())
      {
         // Halb-transparentes Rechteck mit Text zeichnen
         dc.DrawRectangle(new SolidColorBrush(Color.FromArgb(80, 0, 0, 30)),
            null, new Rect(20, 40, this.Width - 40, 40));
         FormattedText formattedText = new FormattedText("Keine Panik",
            Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
            new Typeface("Tahoma"), 28, new SolidColorBrush(
```

Listing 685: Transparentes Zeichnen in einem WPF-Fenster

```
      Color.FromArgb(150, 0, 0, 255)));
   double x = (this.Width - formattedText.Width) / 2;
   double y = 45;
   dc.DrawText(formattedText, new Point(x, y));

   // Bild einlesen
   BitmapDecoder decoder = BitmapDecoder.Create(
      new Uri("Hitchhiker.gif", UriKind.Relative),
      BitmapCreateOptions.None, BitmapCacheOption.None);
   BitmapSource bitmapSource = decoder.Frames[0];

   // Das Bild so umwandeln, dass die Farbe des ersten Pixels
   // voll-transparent und der Rest des Bilds halb-transparent
   // erscheint
   if (bitmapSource.Format != PixelFormats.Bgra32)
   {
      bitmapSource = new FormatConvertedBitmap(bitmapSource,
         PixelFormats.Bgra32, null, 0.0);
   }
   int stride = bitmapSource.PixelWidth *
      ((bitmapSource.Format.BitsPerPixel + 7) / 8);
   byte[] pixelData = new byte[stride * bitmapSource.PixelHeight];
   bitmapSource.CopyPixels(pixelData, stride, 0);
   byte firstPixelBlue = pixelData[0];
   byte firstPixelGreen = pixelData[1];
   byte firstPixelRed = pixelData[2];
   for (int i = 0; i < bitmapSource.PixelHeight * stride;
      i += ((bitmapSource.Format.BitsPerPixel + 7) / 8))
   {
      if (pixelData[i] == firstPixelBlue &&
         pixelData[i + 1] == firstPixelGreen &&
         pixelData[i + 2] == firstPixelRed)
      {
         // Den Alpha-Wert des Pixels auf 0 setzen
         pixelData[i + 3] = 0;
      }
      else
      {
         // Den Alpha-Wert des Pixels auf 160 setzen
         pixelData[i + 3] = 160;
      }
   }

   // Mit den neuen Pixel-Daten ein neues Bild erzeugen
   BitmapSource transformedBitmap = BitmapSource.Create(
```

Listing 685: Transparentes Zeichnen in einem WPF-Fenster (Forts.)

```
            bitmapSource.PixelWidth, bitmapSource.PixelHeight,
            bitmapSource.DpiX, bitmapSource.DpiY, PixelFormats.Bgra32,
            bitmapSource.Palette, pixelData, stride);

         // Das transformierte Bild zeichnen
         Rect bitmapRect = new Rect((this.Width - 20 -
           transformedBitmap.Width) / 2, 55, transformedBitmap.Width,
           transformedBitmap.Height);
         dc.DrawImage(transformedBitmap, bitmapRect);
      }

      // Den DrawingVisual dem Fenster als visuelles und logisches
      // Child-Element hinzufügen
      this.AddVisualChild(this.drawingVisual);
      this.AddLogicalChild(this.drawingVisual);
   }

   /* Wird überschrieben, um die Anzahl der visuellen Kind-Elemente
      um das eigene zu erhöhen */
   protected override int VisualChildrenCount
   {
      get
      {
         return base.VisualChildrenCount + 1;
      }
   }

   /* Wird überschrieben, um das eigene visuelle Kind-Element zurückzugeben */
   protected override Visual GetVisualChild(int index)
   {
      if (index < base.VisualChildrenCount)
      {
         return base.GetVisualChild(index);
      }
      else
      {
         return this.drawingVisual;
      }
   }
}
```

Listing 685: Transparentes Zeichnen in einem WPF-Fenster (Forts.)

Abbildung 187 zeigt das Ergebnis dieses Programms (mit dem Mond als Hintergrundbild).



Abbildung 187: (Halb-)Transparentes Zeichnen eines Rechtecks, eines Textes und eines Bitmaps auf einem Formular mit Hintergrundbild

Windows.Forms

In einer `Windows.Forms`-Anwendung können Sie zum transparenten Zeichnen von Bildern die `MakeTransparent`-Methode eines `Bitmap`-Objekts, ein `ColorMap`-Array und/oder ein `ColorMatrix`-Objekt (aus dem Namensraum `System.Drawing.Imaging`) verwenden (siehe Rezept 334). Die `MakeTransparent`-Methode erläutere ich hier nicht weiter. Dieser Methode übergeben Sie einfach die Farbe, die in Transparenz umgewandelt werden soll.

Über ein `ColorMap`-Array können Sie einzelne Farben in Farben mit einem niedrigen Alphawert umdefinieren. Damit besitzen Sie mehr Flexibilität, weil Sie über den Alphawert der neuen Farbe den Grad der Transparenz bestimmen können. Ein `ColorMatrix`-Objekt ermöglicht die Verminderung des Alphawerts für das gesamte Bild.

In Listing 686 wird auf diese Weise ein Bild, das im Anwendungsordner gespeichert ist, ausgegeben. Der original schwarze Hintergrund des Bilds wird über ein `ColorMap`-Objekt in einen durchsichtigen Hintergrund umgewandelt. Ein `ColorMatrix`-Objekt übernimmt die Transformation des Alphawerts der einzelnen Pixel, so dass dieser nur noch 70 Prozent des originalen Werts beträgt.

Neben den für `Windows.Forms`-Anwendungen üblichen Referenzen und Importen müssen Sie zum Kompilieren dieses Programms die Assembly *System.Drawing.dll* referenzieren und die Namensräume `System.Drawing`, `System.Drawing.Imaging`, `System.IO` und `System.Windows.Forms` importieren. Außerdem muss ein Bild mit Namen *Hitchhiker.gif* im Anwendungsordner gespeichert sein.

```
private void StartForm_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
   // Das Graphics-Objekt referenzieren
   Graphics g = e.Graphics;
```

Listing 686: Transparentes Zeichnen von Grafiken und Bildern in einer Windows.Forms-Anwendung

```
// Halb-transparentes Rechteck mit Text zeichnen
using (Brush brush = new SolidBrush(Color.FromArgb(80, 0, 0, 30)))
{
   g.FillRectangle(brush, 20, 60,
      this.ClientRectangle.Width - 40, 50);
}
using (Font font = new Font("Tahoma", 20, FontStyle.Bold))
{
   using (Brush brush = new SolidBrush(Color.FromArgb(80, 0, 0, 255)))
   {
      SizeF textSize = g.MeasureString("Hitchhiker", font);
      int x = (int)((this.ClientRectangle.Width - textSize.Width) / 2);
      int y = 65;
      g.DrawString("Keine Panik", font, brush, x, y);
   }
}

// ColorMap für die Umwandlung aller schwarzen Pixel in
// durchsichtige Pixel erzeugen
ColorMap[] colorMap = new ColorMap[1];
colorMap[0] = new ColorMap();
colorMap[0].OldColor = Color.Black;
colorMap[0].NewColor = Color.FromArgb(0, 0, 0, 0);

// ColorMatrix für die Transformation des Alphawerts erzeugen
ColorMatrix colorMatrix = new ColorMatrix(new float[][] {
   new float[] {1, 0, 0, 0, 0},
   new float[] {0, 1, 0, 0, 0},
   new float[] {0, 0, 1, 0, 0},
   new float[] {0, 0, 0, 0.7F, 0},
   new float[] {0, 0, 0, 0, 1}
   });

// Neues ImageAttributes-Objekt erzeugen und das ColorMap- und
// ColorMatrix-Objekt übergeben
ImageAttributes imageAttributes = new ImageAttributes();
imageAttributes.SetRemapTable(colorMap);
imageAttributes.SetColorMatrix(colorMatrix);

// Bitmap lesen und zeichnen
string filename = Path.Combine(Application.StartupPath, "Hitchhiker.gif");
using (Bitmap hitchhikerBitmap = new Bitmap(filename))
{
   Rectangle destRect = new Rectangle(
      (int)((this.ClientRectangle.Width - hitchhikerBitmap.Width) / 2),
```

Listing 686: Transparentes Zeichnen von Grafiken und Bildern in einer Windows.Forms-Anwendung (Forts.)

```
            100, hitchhikerBitmap.Width, hitchhikerBitmap.Height);
        g.DrawImage(hitchhikerBitmap, destRect, 0, 0, hitchhikerBitmap.Width,
            hitchhikerBitmap.Height, GraphicsUnit.Pixel,
            imageAttributes);
    }
}
```

Listing 686: Transparentes Zeichnen von Grafiken und Bildern in einer Windows.Forms-Anwendung (Forts.)

343 Bilder mit Schatten zeichnen

Schatten sind in WPF ein sehr einfach zu lösendes Problem, das im einfachsten Fall noch nicht einmal Programmierung erfordert. In `Windows.Forms`-Anwendungen müssen Sie allerdings programmieren.

WPF

In WPF können Sie jedem UI-Element (allen Klassen, die von `UIElement` abgeleitet sind), über die Eigenschaft `BitmapEffect` (u. a.) einen Schatten zuordnen, indem Sie ein `DropShadowBitmapEffect`-Objekt in diese Eigenschaft schreiben. Das geht natürlich direkt in XAML:

```
<Image Margin="10,10,83,123" Name="image1" Stretch="None">
    <Image.Source>
        <BitmapImage UriSource="Hitchhiker.jpg"/>
    </Image.Source>
    <Image.BitmapEffect>
        <DropShadowBitmapEffect Color="Black" Direction="310" Noise="0.1"
            Opacity="0.7" ShadowDepth="10" Softness="0.5" />
    </Image.BitmapEffect>
</Image>
```

Listing 687: Image-Steuerelement mit Schatten-Effekt



Abbildung 188: Ein WPF-Fenster (mit Hintergrund-Textur) mit Image-Steuerelement und Schatten-Effekt

Über die verschiedenen Eigenschaften der `DropShadowBitmapEffect`-Klasse können Sie den Schatten beeinflussen:

- `Color`: Die Farbe des Schattens. Der Standardwert ist Schwarz.
- `Direction`: Der Winkel des Schattens. Der Winkel 0 bedeutet, dass der Schatten auf der rechten Seite des UI-Elements liegt. Die Winkelangabe ist entgegen des Uhrzeigersinns! Der Standardwert ist 315.
- `Noise`: Die Körnung des Schattens (der Rauschpegel) mit einem Wert zwischen 0 (kein Rauschen) und 1 (maximales Rauschen). Der Defaultwert ist 0.
- `Opacity`: Die Durchsichtigkeit des Schattens mit einem Wert zwischen 0 (vollkommen transparent) und 1 (vollkommen undurchsichtig). Der Defaultwert ist 1.
- `ShadowDepth`: Die Breite des Schattens mit Werten zwischen 0 und 300. Der Defaultwert ist 5.
- `Softness`: Die Weichzeichnung des Schattens mit einem Wert zwischen 0 (vollkommen scharf) und 1 (maximale Weichzeichnung). Der Defaultwert ist 0,5.

Sie können in WPF aber auch mit Schatten zeichnen. Dazu können Sie zum einen dem `DrawingVisual`-Objekt, auf dem Sie zeichnen, über dessen `BitmapEffect`-Eigenschaft eine Instanz der `DropShadowBitmapEffect`-Klasse zuweisen, die Sie Ihren Anforderungen entsprechend initialisiert haben. Zum anderen können Sie dem `DrawingContext` über dessen `PushEffect`-Methode eine `DropShadowBitmapEffect`-Instanz zuweisen. Effekte, die über `PushEffect` hinzugefügt wurden, bleiben für alle folgenden Zeichenoperationen erhalten, bis die `Pop`-Methode aufgerufen wird, die Effekte (und Transformationen) wieder entfernt.

Das folgende Beispiel zeigt das Zeichnen eines Bilds, das als Ressource in der Anwendung gespeichert ist, an dem kompletten Quellcode eines (leeren) Fensters. Der wesentliche Programmcode ist das Erzeugen, Initialisieren und Zuweisen des `DropShadowBitmapEffect` beim Zeichnen im Konstruktor. Der Effekt ist genau derselbe wie in Abbildung 188.

Das Beispiel erfordert, dass eine Bild-Ressource mit dem Namen *Hitchhiker.jpg* in der Anwendung gespeichert ist. Es erfordert außerdem den Import der Namensräume `System`, `System.Windows`, `System.Windows.Media`, `System.Windows.Media.Effects` und `System.Windows.Media.Imaging`.

```
public partial class StartWindow : Window
{
   /* DrawingVisual erzeugen, auf dem gezeichnet werden kann */
   private DrawingVisual drawingVisual = new DrawingVisual();

   /* Konstruktor. Erzeugt die Steuerelemente und Komponenten. */
   public StartWindow()
   {
      InitializeComponent();

      // Den DrawingContext für den DrawingVisual holen
      using (DrawingContext dc = this.drawingVisual.RenderOpen())
```

Listing 688: Zeichnen eines Bilds mit Schatten-Effekt in WPF

```
    {
        // Das Bild aus der Ressource lesen
        BitmapDecoder decoder = BitmapDecoder.Create(
            new Uri("pack://application:,,,/Hitchhiker.jpg"),
            BitmapCreateOptions.None, BitmapCacheOption.None);
        BitmapSource bitmapSource = decoder.Frames[0];

        // Den DropShadowBitmapEffect erzeugen und initialisieren
        DropShadowBitmapEffect effect = new DropShadowBitmapEffect {
            Direction = 310, Noise = 0.1, Opacity = 0.7, ShadowDepth = 10,
            Softness = 0.5 };

        // Den DropShadowBitmapEffect dem DrawingContext hinzufügen
        dc.PushEffect(effect, null);

        // Das Bild zeichnen
        dc.DrawImage(bitmapSource, new Rect(200, 10,
            bitmapSource.Width, bitmapSource.Height));

        // Den Effekt wieder entfernen
        dc.Pop();
    }

    // Den DrawingVisual dem Fenster als visuelles und logisches
    // Child-Element hinzufügen
    this.AddVisualChild(this.drawingVisual);
    this.AddLogicalChild(this.drawingVisual);
}

/* Wird überschrieben, um die Anzahl der visuellen Kind-Elemente
   um das eigene zu erhöhen  */
protected override int VisualChildrenCount
{
    get
    {
        return base.VisualChildrenCount + 1;
    }
}

/* Wird überschrieben, um das eigene visuelle Kind-Element zurückzugeben */
protected override Visual GetVisualChild(int index)
{
    if (index < base.VisualChildrenCount)
    {
        return base.GetVisualChild(index);
```

Listing 688: Zeichnen eines Bilds mit Schatten-Effekt in WPF (Forts.)

```
      }
      else
      {
         return this.drawingVisual;
      }
   }
}
```

Listing 688: Zeichnen eines Bilds mit Schatten-Effekt in WPF (Forts.)

Windows.Forms

In `Windows.Forms`-Anwendungen ist das übliche Verfahren zum Zeichnen eines Bilds mit einem Schatten, dass unter dem Bild ein gefülltes Rechteck gezeichnet wird, das um einige Pixel nach rechts und unten verschoben ist. Wenn Sie für die Farbe des Rechtecks Schwarz mit einem Alphawert kleiner 255 verwenden, wird der Schatten leicht durchsichtig und damit mehr oder weniger realistisch (im Vergleich zu den realistischeren WPF-Schatten).

Die Methode `DrawBitmapWithShadow` in Listing 689 macht genau das. Das Rechteck unter dem Bild wird mit einem Alphawert von 160 gezeichnet, was eine leichte Durchsichtigkeit bewirkt. Damit die Schattengröße einstellbar ist, erwartet `DrawBitmapWithShadow` neben dem `Graphics`-Objekt, auf dem gezeichnet werden soll, dem Bild und der Ausgabeposition auch einen Wert für die Schattengröße.

Diese Methode benötigt eine Referenz auf die Assembly *System.Drawing.dll* und den Import der Namensräume `System` und `System.Drawing`.

```
public static void DrawBitmapWithShadow(Graphics g, Image image,
   int left, int top, int shadowSize)
{
   // Schatten zeichnen
   Color shadowColor = Color.FromArgb(160, 0, 0, 0);
   g.FillRectangle(new SolidBrush(shadowColor), left + shadowSize,
      top + shadowSize, image.Width, image.Height);

   // Bild zeichnen
   g.DrawImage(image, new Rectangle(left, top, image.Width, image.Height),
      0, 0, image.Width, image.Height, GraphicsUnit.Pixel);
}
```

Listing 689: Methode zum Zeichnen eines Bilds mit einem Schatten unter GDI

344 Schräg zeichnen und Zeichenobjekte rotieren

Das schräge Zeichnen (bzw. in WPF auch das schräge Ausgeben) von (Zeichen-)Objekten ist über eine Transformation recht einfach. Eine Transformation habe ich bereits beim Zeichnen eines Pfeils eingesetzt (siehe Rezept 341). Eine Transformation bewirkt, dass die Originalkoordinaten eines zu zeichnenden Objekts vor den Ausgaben transformiert werden. Sie können

eine Verschiebung (in X/Y-Richtung), eine Rotation in einem Drehwinkel, eine Skalierung und ein Kippen des Objekts erreichen.

WPF

In WPF können Sie wie auch beim Anwenden von Effekten Transformationen auf allen UI-Elementen über deren `LayoutTransform`-Eigenschaft anwenden. Sie können Transformationen aber auch beim Zeichnen anwenden, wie ich dies in Rezept 341 gemacht habe, um den gezeichneten Pfeil zu drehen und zu verschieben.

Zum Transformieren stehen Ihnen die folgenden Klassen (aus dem Namensraum `System.Windows.Media`) zur Verfügung:

- `MatrixTransform`: Ermöglicht eine freie Transformation mit eigenen Matrix-Daten, die einem `Matrix`-Objekt entstammen
- `RotateTransform`: Rotiert ein Objekt um einen definierten Rotationspunkt mit einem angegebenen Winkel
- `ScaleTransform`: Skaliert ein Objekt
- `SkewTransform`: Ermöglicht eine zweidimensionale Neigung
- `TranslateTransform`: Ermöglicht das Verschieben eines Objekts auf der X- und der Y-Achse
- `TransformGroup`: Über ein solches Objekt können Sie mehrere Transformationen auf einem Objekt anwenden

In XAML können Sie Transformationen über die `LayoutTransform`-Eigenschaft auch deklarativ anwenden. Das folgende Beispiel deklariert ein Label, das um 270 Grad gedreht wird, ein Rechteck, das um 30 Grad gedreht wird, und ein Rechteck, das um 30 Grad in X-Richtung gekippt und auf den Faktor 0,8 skaliert wird:

```
<!-- Ein um 270 Grad gedrehtes Label -->
<Label Height="28" HorizontalAlignment="Left" Margin="10,10,0,0"
  Name="rotateTransformDemoLabel" VerticalAlignment="Top"
  Content="Das ist ein in XAML gedrehter Text">
    <Label.LayoutTransform>
        <RotateTransform Angle="270"/>
    </Label.LayoutTransform>
</Label>

<!-- Ein um 30 Grad gedrehtes Rechteck -->
<Rectangle Width="100" Height="100" Fill="Red" Stroke="Navy" HorizontalAlignment="Left"
   Margin="50,15,0,0" VerticalAlignment="Top">
    <Rectangle.LayoutTransform>
        <RotateTransform Angle="30"/>
    </Rectangle.LayoutTransform>
</Rectangle>
```

Listing 690: Transformationen von UI-Elementen in XAML

```
<!-- Ein um 30 Grad in X-Richtung gekipptes Rechteck -->
<Rectangle Width="100" Height="100" Fill="Red" Stroke="Navy" HorizontalAlignment="Left"
   Margin="200,15,0,0" VerticalAlignment="Top">
    <Rectangle.LayoutTransform>
        <TransformGroup>
            <!-- Kipp-Transformation -->
            <SkewTransform AngleX="30"/>
            <!-- Skalierungs-Transformation -->
            <ScaleTransform ScaleX="0.8" ScaleY="0.8"/>
        </TransformGroup>
    </Rectangle.LayoutTransform>
</Rectangle>
```

Listing 690: Transformationen von UI-Elementen in XAML (Forts.)

Das Ergebnis lässt sich am besten im Designer beurteilen, weil dieser die Fläche, die die Objekte ohne Transformation ausfüllen würden, zusätzlich zu den transformierten Objekten darstellt (Abbildung 189).

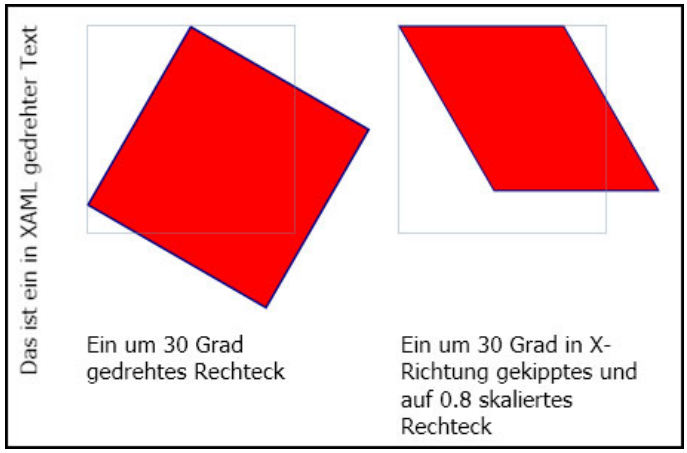


Abbildung 189: Das XAML-Transformations-Beispiel im Designer

Transformationen können Sie natürlich auch beim eigenen Zeichnen anwenden. Dazu können Sie einmal die `Transform`-Eigenschaft des `DrawingVisual` mit einer Transformations-Instanz (oder mehreren in einer `TransformGroup`-Instanz) belegen, um den gesamten Visual (auf dem Sie zeichnen) zu transformieren.

Wenn Sie beim Zeichnen allerdings dynamisch transformieren wollen, setzen Sie idealerweise die `PushTransform`-Methode des `DrawingContext` ein, auf dem Sie zeichnen. Diese Methode können Sie mehrfach aufrufen, um mehrere Transformationen zu definieren. Diese Transformationen werden auf alle nachfolgenden Zeichenoperationen angewendet, bis Sie Transformationen über die `Pop`-Methode wieder entfernen! So können Sie recht einfach Objekte verdrehen, skalieren, kippen und verschieben.

Beim Verdrehen und Kippen von Objekten ist der Rotationsmittelpunkt wichtig. Dieser befindet sich per Voreinstellung an der Position 0,0, also der linken oberen Ecke des `DrawingVisual`, auf dem Sie zeichnen. Objekte werden aber meist um ihre linke obere Ecke oder um ihren Mittelpunkt gedreht oder gekippt. Deshalb sollten Sie den Drehpunkt über die Eigenschaften `CenterX` und `CenterY` des Transformationsobjekts (über den Konstruktor) entsprechend festlegen.

Das folgende Beispiel zeichnet zunächst einen Text um 45 Grad gedreht. Als Drehpunkt wird die linke obere Ecke des Textes verwendet. Danach wird ein Rechteck um 45 Grad gedreht gezeichnet, dessen Drehpunkt auf die Mitte des Rechtecks gelegt wird. Schließlich wird dasselbe Rechteck noch einmal normal (nicht gedreht) ausgegeben.

Das Programm läuft in einem WPF-Fenster, das ein privates Feld `drawingVisual` vom Typ `DrawingVisual` enthält (und das entsprechend zum Zeichnen eingerichtet ist).

```
// Den DrawingContext für den DrawingVisual holen
using (DrawingContext dc = this.drawingVisual.RenderOpen())
{
   // Einen Text um 45 Grad verdreht ausgeben. Der Drehpunkt
   // wird dabei auf den Startpunkt des Textes gesetzt.
   FormattedText formattedText = new FormattedText(
      "Das ist selbst gezeichneter, gedrehter Text",
      Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
      new Typeface("Tahoma"), 14, Brushes.Black);
   dc.PushTransform(new RotateTransform(45, 350, 10));
   dc.DrawText(formattedText, new Point(350, 10));
   dc.Pop(); // Transformationen und Effekte wieder entfernen

   // Ein Rechteck um 45 Grad nach rechts gekippt zeichnen.
   // Als Drehpunkt wird die Mitte des Rechtecks angegeben.
   dc.PushTransform(new RotateTransform(45, 455, 105));
   dc.DrawRectangle(Brushes.Navy, new Pen(Brushes.Red, 2),
      new Rect(450, 100, 10, 10));
   dc.Pop(); // Transformationen und Effekte wieder entfernen

   // Normales Rechteck zeichnen
   dc.DrawRectangle(Brushes.Navy, new Pen(Brushes.Red, 2),
      new Rect(450, 100, 10, 10));
}
```

Listing 691: Transformationen beim Zeichnen über PushTransform

Abbildung 190 zeigt einen Ausschnitt der Beispielanwendung, auf der dieser Code angewendet wurde.

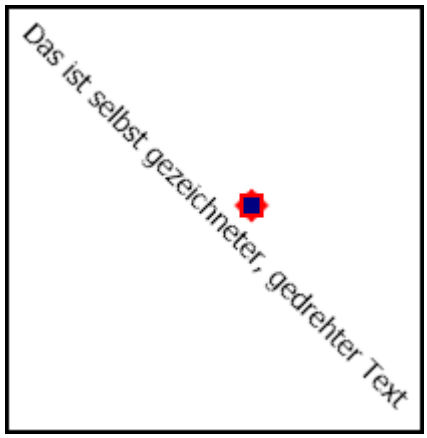


Abbildung 190: Das Beispiel zum Transformieren beim Zeichnen in Aktion

Hinweis

Das Drehen von Text ist leider gar nicht so einfach. In Rezept 346 finden Sie eine Lösung für das in 90-Grad-Schritten gedrehte Zeichen an einer definierten Position.

Damit sind die Möglichkeiten aber noch nicht erschöpft, denn Sie können auch einige Objekte, die Sie zum Zeichnen verwenden, wie z. B. ein `Geometry`-Objekt, über deren `Transform`-Eigenschaft separat transformieren. Andere Typen, wie z. B. die `Rect`-Struktur, besitzen die Möglichkeit, über ein `Matrix`-Objekt transformiert zu werden. Ein `Matrix`-Objekt verwaltet die Daten einer oder mehrerer Transformationen. Über die Methode `Translate` können Sie eine Verschiebung in X/Y-Richtung bewirken, die Methoden `Rotate` und `RotateAt` führen zu einer Rotation. `Rotate` dreht dabei um die linke obere Ecke, bei `RotateAt` können Sie den Drehpunkt bestimmen. Ein `Matrix`-Objekt habe ich mit einer `MatrixTransformation` auf einem `StreamGeometry`-Objekt in meinem Pfeil-Rezept eingesetzt (Rezept 341). Vergleichen Sie dieses, um zu erfahren, wie Sie dies programmieren.

Windows.Forms

`Windows.Forms` kennt keine speziellen Transformations-Objekte wie WPF, aber eine `Matrix`, die der in WPF ähnlich ist. Eine `Graphics`-Instanz verwaltet eine solche in der Eigenschaft `Transform`. In diese Eigenschaft können Sie ein neu erzeugtes und initialisiertes `Matrix`-Objekt schreiben um eine spezielle Transformation zu erreichen (der direkte Zugriff auf das `Matrix`-Objekt der `Graphics`-Instanz führt leider zu keinem Resultat). Wie bei WPF bewirkt die Methode `Translate` eine Verschiebung in X/Y-Richtung, die Methoden `Rotate` und `RotateAt` bewirken eine Verdrehung. `Rotate` dreht (ebenfalls wie bei WPF) um die linke obere Ecke, bei `RotateAt` können Sie den Drehpunkt bestimmen.

An Stelle eines eigenen `Matrix`-Objekts können Sie aber auch Methoden des `Graphics`-Objekts direkt verwenden (allerdings etwas eingeschränkt). Die Methode `TranslateTransform` stellt eine Transformation der Zeichenmatrix in X/Y-Richtung (also eine Verschiebung des Ursprungs des

Koodinatensystems) ein, RotateTransform initiiert eine Drehung um einen bestimmten Winkel mit dem Ursprung des Koodinatensystems als Mittelpunkt.

Wichtig bei GDI-Transformationen ist das zweite Argument (order) der verwendeten Methoden. Dieses Argument bestimmt, wann welche Transformation angewendet wird. Mit dem Wert MatrixOrder.Prepend wird die Transformation vor der vorherigen angewendet, mit dem Wert MatrixOrder.Append nach der vorherigen. Dieses Argument ist zur korrekten Anwendung der Transformation (in GDI) sehr wichtig, wenn Sie mehrfach zeichnen, z. B. ein Rechteck um definierte X/Y-Werte verschoben und mehrfach um jeweils 30 Grad weiterverdreht. Da jeder Aufruf einer Transformationsmethode die Transformation relativ zur aktuellen bewirkt, können Sie Translate bzw. TranslateTransform hier nur einmal aufrufen. Vor dem Zeichnen der Rechtecke rufen Sie dann Rotate, RotateAt bzw. RotateTransform auf, um die Drehung um weitere 30 Grad fortzuführen. Dabei müssen Sie mit MatrixOrder.Prepend angeben, dass die vorherige Transformation (die Verschiebung) vor der Verdrehung angewendet wird, wenn der Mittelpunkt der einzelnen Drehungen derselbe sein soll.

Listing 692 zeigt neben dem normalen Zeichnen zunächst das schräge und versetzte Zeichnen eines Textes. Dabei wird die Transformation über TranslateTransform so eingestellt, dass die Koordinaten um 20 Punkte in X- und um 25 Punkte in Y-Richtung verschoben werden. Über RotateTransform wird eine Rotation um 30 Grad erreicht. Bei der nachfolgenden Ausgabe eines Textes wird diese Transformation angewendet. Das Beispiel setzt die Transformation über ResetTransform dann wieder zurück und zeichnet einen geraden Text.

Zum Kompilieren benötigen Sie ein Formular, dessen Paint-Ereignis der Beispielmethode zugewiesen ist, die üblichen Referenzen und Importe und den Import der Namensräume System.Drawing und System.Drawing.Drawing2D.

```
private void StartForm_Paint(object sender, PaintEventArgs e)
{
   // Graphics-Objekt referenzieren
   Graphics g = e.Graphics;

   // SmoothingMode auf HighQuality einstellen, damit die Linien sauber
   // gezeichnet werden
   g.SmoothingMode = SmoothingMode.HighQuality;

   // Zeichenmatrix so verschieben, dass die nachfolgenden Zeichnungen
   // um 20 Pixel in X- und um 25 Pixel in Y-Richtung versetzt
   // und um 30 Grad gedreht erscheinen
   g.TranslateTransform(20, 25, MatrixOrder.Prepend);
   g.RotateTransform(30, MatrixOrder.Prepend);

   // Text schräg ausgeben
   g.DrawString("Das ist ein gedrehter Text", new Font("Arial", 14),
      new SolidBrush(Color.Black), 0, 0);
```

Listing 692: Schräges Zeichnen eines Textes

```
// Transformation wieder zurücksetzen
g.ResetTransform();

// Text gerade ausgeben
g.DrawString("Das ist ein gerader Text", new Font("Arial", 14),
   new SolidBrush(Color.Black), 0, 0);
```

Listing 692: Schräges Zeichnen eines Textes (Forts.)

Drehungen werden immer über einen Drehpunkt ausgeführt. Per Default ist das der linke obere Eckpunkt der aktuellen Zeichenmatrix (die eventuell über vorherige Transformationen bereits transformiert wurde). Über eine gezielte Verschiebung mit nachfolgender Drehung können Sie Ausgaben recht flexibel verdrehen. Für einige Aufgabenstellungen müssen Sie aber um einen anderen Mittelpunkt drehen. Dazu verwenden Sie dann ein `Matrix`-Objekt und dessen `RotateAt`-Methode, der Sie neben dem Winkel auch den Mittelpunkt der Drehung übergeben können. Das `Matrix`-Objekt weisen Sie der `Transform`-Eigenschaft des `Graphics`-Objekts zu, um die Transformation zu definieren.

Hinweis

Bei der Verwendung der `Transform`-Eigenschaft sollten Sie beachten, dass Sie dieser immer ein neues `Matrix`-Objekt zuweisen müssen, wenn Sie eine Transformation anwenden wollen. Die Verwendung des Objekts, das `Transform` referenziert, erbringt (eigenartigerweise) keinen Erfolg.

Das Beispiel setzt die Transformation schließlich wieder zurück und zeichnet das Rechteck noch einmal normal.

```
// Matrix-Objekt für eine Transformation erstellen und initialisieren
using (Matrix matrix = new Matrix())
{
   // Rechteck ermitteln, das gezeichnet werden soll
   Rectangle rectangle = new Rectangle(250, 50, 100, 100);

   // Transformation so einstellen, dass eine Drehung des Rechtecks
   // um den Mittelpunkt und um 45 Grad erfolgt
   matrix.RotateAt(45, new Point(rectangle.X + (rectangle.Width / 2),
      rectangle.Y + (rectangle.Height / 2)), MatrixOrder.Prepend);

   // Die neue Transformationsmatrix zuweisen
   g.Transform = matrix;

   // Das Rechteck zeichnen
   g.FillRectangle(new SolidBrush(Color.Blue), rectangle);
   g.DrawRectangle(new Pen(new SolidBrush(Color.Red), 1), rectangle);
```

Listing 693: Gedrehtes und normales Zeichnen eines Rechtecks

```
        // Transformation wieder entfernen
        g.ResetTransform();

        // Das Rechteck ohne Transformation noch einmal zeichnen
        g.FillRectangle(new SolidBrush(Color.Red), rectangle);
        g.DrawRectangle(new Pen(new SolidBrush(Color.Blue), 1), rectangle);
    }
}
```

Listing 693: Gedrehtes und normales Zeichnen eines Rechtecks (Forts.)

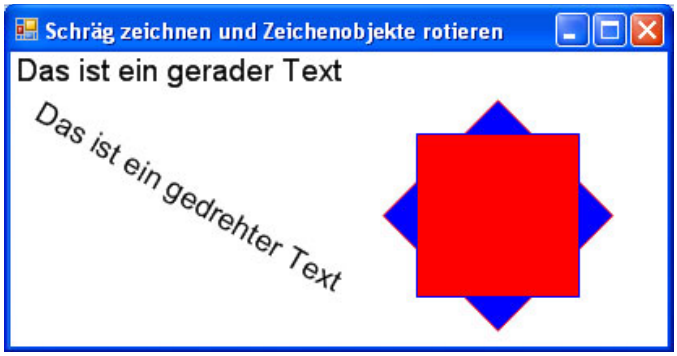


Abbildung 191: Schräges Zeichnen in einer Windows.Forms-Anwendung

345 Den Drehpunkt eines Rechtecks so ermitteln, dass die Ecke links oben an derselben Position bleibt

In einem meiner (`Windows.Forms`-)Projekte habe ich eine Komponente entwickelt, die ein Bild darstellt und das Zeichnen von einigen grafischen Objekten über dem Bild ermöglicht. Um ein Rückgängig-Machen zu erlauben, habe ich die zu zeichnenden Objekte in einer Auflistung verwaltet und im `Paint`-Ereignis der Komponente über das Bild gezeichnet. Das ist soweit nichts Besonderes und kommt wahrscheinlich in jedem Zeichenprogramm vor.

Problematisch wurde das Ganze aber, als ermöglicht werden sollte, dass das Bild um 90 Grad nach rechts oder links gedreht werden kann. Die gezeichneten Objekte mussten dann natürlich mitgedreht werden. Wie Sie in Rezept 344 erfahren haben, können Sie dazu ein `Matrix`-Objekt verwenden. Das (zugegebenermaßen aus meiner jetzigen Sicht eher triviale) Problem ist lediglich, den passenden Drehpunkt zu finden. Denn dieser muss genau die Position besitzen, die dazu führt, dass das Rechteck des Bilds so gedreht wird, dass die obere linke Ecke erhalten bleibt.

Abbildung 192 und Abbildung 193 zeigen, was ich meine. In diesem Programm wurden ein Bild und ein Dreieck gezeichnet. In Abbildung 193 wurden das Bild und das Dreieck um 90 Grad gedreht.



Abbildung 192: Bild mit darüber gezeichnetem Dreieck



Abbildung 193: Um 90 Grad gedrehtes Bild mit darüber gezeichnetem Dreieck

Wie bereits gesagt: Die Lösung dieses Problems ist eher trivial (wenn man weiß, wie es geht ☺). Aber trotzdem haben alle Entwickler, mit denen ich darüber sprach, diese nicht sofort vor Augen gehabt. Dazu muss man wohl Mathematiker sein.

Um den Drehpunkt eines Rechtecks zu ermitteln, der dazu führt, dass die linke obere Ecke des Rechtecks erhalten bleibt, müssen Sie dieses lediglich in ein Quadrat umrechnen. Der Mittelpunkt dieses Quadrats ist dann der Drehpunkt.

Die folgende Methode, bei der das einzig Komplizierte der Name ist (nach dem ich Stunden gesucht habe ☺), macht im Prinzip genau das. Diese Methode berücksichtigt sogar, ob das Rechteck nach links oder nach rechts gedreht wird. Ich stelle hier nur die WPF-Version vor, die nahezu identische GDI-Version finden Sie im GDI-Beispiel zu diesem Rezept, in der Codebook-Klassenbibliothek und im Repository.

Zum Kompilieren müssen Sie den Namensraum `System.Windows` einbinden.

```
public static Point
   GetCenterOfRotationForRotationWithRetentionOfUpperLeftAngle(
   Rect rect, bool rotateClockwise)
{
   // Rotier-Punkt berechnen
   double offset = 0;
   if (rotateClockwise)
   {
      offset = rect.Height / 2;
   }
   else
   {
      offset = rect.Width / 2;
   }

   // Ergebnis zurückgeben
   return new Point(rect.X + offset, rect.Y + offset);
}
```

Listing 694: WPF-Version der Methode zur Berechnung des Rotationspunkts eines Rechtecks für eine Rotation um 90 Grad unter Beibehaltung der linken oberen Ecke

Die Beispiele zu diesem Rezept erläutern die Anwendung dieser Methode.

346 Text an einer definierten Position in 90-Grad-Schritten gedreht ausgeben

Text gedreht an einer bestimmten Position auszugeben ist gar nicht so einfach. Das Problem ist, dass Sie den Drehpunkt korrekt berechnen müssen (vergleichen Sie dazu das Rezept 344). Ich habe allerdings eine Lösung für das Drehen von Text in 90-Grad-Schritten gefunden. Die (Erweiterungs-)Methode `DrawRotatedText` in Listing 695 berechnet dazu zuerst das Rechteck, das den normal gezeichneten Text umfasst. Für dieses Rechteck ermittelt `DrawRotatedText` dann nach der Technik aus Rezept 345 den Drehpunkt, der das Rechteck so drehen würde, dass die linke, obere Ecke bestehen bleibt. Bei einer 180-Grad-Drehung ist der Drehpunkt allerdings einfach die Mitte des Rechtecks. Dieser Drehpunkt wird dann für eine Transformation verwendet, bevor der Text gezeichnet wird. Listing 695 zeigt die WPF-Version von `DrawRotatedText`. Die GDI-Version finden Sie im GDI-Beispiel zu diesem Rezept, in der Codebook-Klassenbibliothek und im Repository.

Zum Kompilieren des Quelltexts müssen Sie die Namensräume `System.Windows`, `System.Windows.Media` und `System.Windows.Media.Imaging` einbinden.

```
public static void DrawRotatedText(this DrawingContext dc,
   FormattedText formattedText, Point origin, Rotation rotation)
{
   // Rechteck berechnen, das den Text umfasst
   Rect textRect = new Rect(origin.X, origin.Y, formattedText.Width,
      formattedText.Height);

   // Drehpunkt so berechnen, dass das Rechteck
   // an der angegebenen Position bleibt
   Point rotationPoint = new Point();
   float rotationAngle = 0;
   double offset = 0;
   switch (rotation)
   {
      case Rotation.Rotate0:
         // Den Text direkt ausgeben
         dc.DrawText(formattedText, origin);
         return;

      case Rotation.Rotate90:
         offset = textRect.Height / 2;
         rotationPoint.X = textRect.X + offset;
         rotationPoint.Y = textRect.Y + offset;
         rotationAngle = 90;
         break;

      case Rotation.Rotate180:
         // Bei einer Drehung um 180 Grad liegt der Drehpunkt in der
         // Mitte des Rechtecks
         rotationPoint.X = textRect.X + (textRect.Width / 2);
         rotationPoint.Y = textRect.Y + (textRect.Height / 2);
         rotationAngle = 180;
         break;

      case Rotation.Rotate270:
         offset = textRect.Width / 2;
         rotationPoint.X = textRect.X + offset;
         rotationPoint.Y = textRect.Y + offset;
         rotationAngle = 270;
         break;
   }

   // Text gedreht ausgeben
   dc.PushTransform(new RotateTransform(rotationAngle,
      rotationPoint.X, rotationPoint.Y));
```

Listing 695: Methode zum gedrehten Zeichnen von Text unter WPF

```
    dc.DrawText(formattedText, origin);
    dc.Pop();
}
```

Listing 695: Methode zum gedrehten Zeichnen von Text unter WPF (Forts.)

Abbildung 194 zeigt eine Anwendung, bei der ein zweizeiliger Text mit Hilfe der `DrawRotatedText`-Methode gedreht ausgegeben wurde. Die Punkte symbolisieren die X/Y-Koordinate der einzelnen Texte.

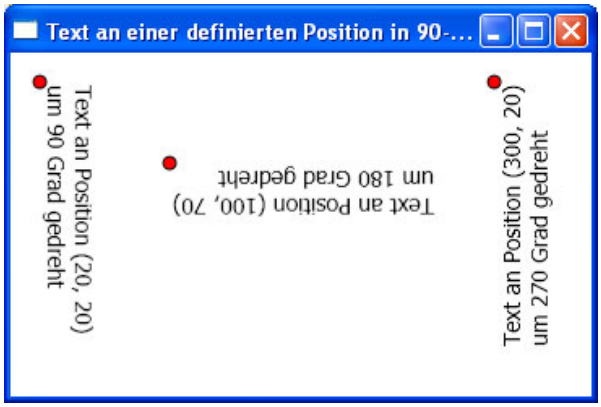


Abbildung 194: Eine Beispielanwendung hat Text gedreht ausgegeben

347 Die Breite und Höhe eines auszugebenden Textes bestimmen

Bei der Ausgabe von Texten ist es häufig wichtig zu wissen, wie breit und hoch ein Text bei der Ausgabe in einer definierten Schriftart erscheint. So können Sie einen Text z. B. in der Mitte eines Formulars ausgeben (wozu Sie allerdings auch die Technik verwenden können, die ich in Rezept 348 zeige).

WPF

In WPF ist diese Aufgabe einfach zu lösen, da Sie zum Zeichnen von Texten ein `FormattedText`-Objekt verwenden. Die Eigenschaft `Width` liefert die Breite des Textes, die Eigenschaft `Height` die Höhe. Damit können Sie z. B. mittig auf einem Fenster zeichnen.

Das folgende Beispiel läuft in einem WPF-Fenster, das ein privates Feld `drawingVisual` vom Typ `DrawingVisual` enthält (und das entsprechend zum Zeichnen eingerichtet ist, siehe in Listing 688 in Rezept 343).

```
// Den DrawingContext für den DrawingVisual holen
using (DrawingContext dc = this.drawingVisual.RenderOpen())
{
```

Listing 696: Mittiges Zeichnen auf einem WPF-Fenster

```
    // Einen Text zentriert ausgeben
    FormattedText formattedText = new FormattedText("Zentrierter Text",
        Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
        new Typeface("Tahoma"), 14, Brushes.Black);
    double x = (this.Width - SystemParameters.ResizeFrameVerticalBorderWidth -
        formattedText.Width) / 2;
    double y = (this.Height - SystemParameters.WindowCaptionHeight -
        SystemParameters.ResizeFrameHorizontalBorderHeight -
        formattedText.Height) / 2;
    dc.DrawText(formattedText, new Point(x, y));
}
```

Listing 696: Mittiges Zeichnen auf einem WPF-Fenster (Forts.)

Windows.Forms

In einer `Windows.Forms`-Anwendung können Sie zur Ermittlung der Breite und der Höhe eines Textes können die Methode `MeasureString` eines `Graphics`-Objekts verwenden. Am ersten Argument übergeben Sie den String, am zweiten ein `Font`-Objekt, das die Schriftart bestimmt. `MeasureString` liefert ein `SizeF`-Objekt zurück, aus dessen Eigenschaften `Width` und `Height` Sie die Breite und Höhe des Textes auslesen können.

`MeasureString` führt aber manchmal zu Problemen. Einmal werden per Voreinstellung rechte Leerzeichen nicht mit eingerechnet (linke schon). Hinzu kommt, dass `MeasureString` einen kleinen Rand zum ermittelten Wert addiert, der bewirken soll, dass Ausgaben auf gerasterten Flächen korrekt erfolgen (was auch prinzipiell funktioniert). Das letzte Problem ist, dass `MeasureString` in der Voreinstellung die Breite von Texten nicht korrekt berechnet, wenn diese verschiedene Satzzeichen wie z. B. Doppelpunkte beinhalten.

Sie können die Einstellungen, die für die Berechnung verwendet werden, aber auch anpassen. Dazu setzen Sie ein `StringFormat`-Objekt ein (aus dem Namensraum `System.Drawing`). Ein solches Objekt verwaltet Informationen zum Textlayout. Die `StringFormat`-Klasse bietet dazu zwei statische Eigenschaften `GenericTypographic` und `GenericDefault`, über die Sie ein solches Objekt erzeugen können. Ein Default-`StringFormat`-Objekt definiert zunächst das Default-Verhalten (mit Randzugabe), ein typografisches liefert echte Werte ohne Zugaben. Über verschiedene Eigenschaften können Sie die Berechnung der Werte beeinflussen. Die Eigenschaft `FormatFlags` verwaltet z. B. Flags für die Formatierung. Das Flag `StringFormatFlags.MeasureTrailingSpaces` bewirkt, dass anhängende Leerzeichen mit eingerechnet werden.

Das Problem der inkorrekten Berechnung der Breite von Strings mit bestimmten Satzzeichen können Sie lösen, indem Sie ein typografisches `StringFormat`-Objekt bei der Berechnung übergeben. Dabei ist es sehr wichtig, dass Sie dieses Objekt ebenfalls der `DrawString`-Methode am Argument `format` übergeben, damit diese dieselben Layoutinformationen verwendet, die bei der Berechnung eingesetzt wurden.

Die Methode `StartForm_Paint` in Listing 697 zeigt, wie Sie mit `MeasureString` umgehen. `StartForm_Paint` (die dem `Paint`-Ereignis eines Formulars zugewiesen ist) zeichnet zuerst einen Text, der vorne und hinten Leerzeichen enthält, mittig auf dem Formular. Um die rechten Leerzeichen mit in der Berechnung zu berücksichtigen, erzeugt unwd initialisiert die Methode

ein neues Default-StringFormat-Objekt mit dem Flag StringFormatFlags.MeasureTrailingSpaces. Beim Aufruf der MeasureString-Methode wird am dritten Argument -1 übergeben. An diesem Argument können Sie eine Maximalbreite übergeben, -1 steht dafür, dass keine Maximalbreite definiert ist. Am letzten Argument übergibt StartForm_Paint das StringFormat-Objekt.

Danach zeichnet das Beispiel denselben Text um eine Texthöhe nach unten versetzt, verwendet dieses Mal aber ein typografisches StringFormat-Objekt. Den Unterschied der beiden Varianten sehen Sie in Abbildung 195.

Dieses Beispiel benötigt eine Referenz auf die Assembly *System.Drawing.dll*, den Import des Namensraums System.Drawing und natürlich ein Formular, dessen Paint-Ereignis mit dieser Methode initialisiert wurde.

```
private void StartForm_Paint(object sender, PaintEventArgs e)
{
   Graphics g = e.Graphics;

   // Ermitteln der Breite und der Höhe des auszugebenden Textes
   string text = "   Test1::::Test2.Test3   ";
   using (Font font = new Font("Tahoma", 20, FontStyle.Bold))
   {
      using (Brush brush = new SolidBrush(Color.Black))
      {
         StringFormat sf = StringFormat.GenericDefault;
         sf.FormatFlags = StringFormatFlags.MeasureTrailingSpaces;
         SizeF textSize = g.MeasureString(text, font, -1, sf);

         // Text mittig und etwas nach oben versetzt ausgeben
         float x = (this.ClientRectangle.Width - textSize.Width) / 2;
         float y = (this.ClientRectangle.Height - textSize.Height) / 2;
         y -= textSize.Height / 2;
         g.DrawString(text, font, brush, x, y);

         // Noch einmal mit einem typografischen StringFormat-Objekt
         sf = StringFormat.GenericTypographic;
         sf.FormatFlags = StringFormatFlags.MeasureTrailingSpaces;
         textSize = g.MeasureString(text, font, -1, sf);
         x = (this.ClientRectangle.Width - textSize.Width) / 2;
         y = ((this.ClientRectangle.Height - textSize.Height) / 2);
         y += textSize.Height / 2;
         g.DrawString(text, font, brush, x, y, sf);
      }
   }
}
```

Listing 697: Berechnen der Breite und Höhe eines auszugebenden Textes zur mittigen Ausgabe

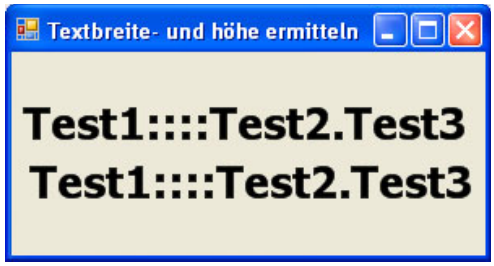


Abbildung 195: Mittige Ausgabe zweier Texte in einer Windows.Forms-Anwendung. Die Position des oberen wurde über ein Default-StringFormat-Objekt, die des unteren über ein typografisches StringFormat-Objekt berechnet

348 Texte zentriert oder rechtsbündig zeichnen

Texte können Sie beim Zeichnen relativ einfach zentriert oder rechtsbündig ausgeben. Die Technik dazu unterscheidet sich in WPF und `Windows.Forms`.

WPF

In einer WPF-Anwendung setzen Sie ein `FormattedText`-Objekt zum Zeichnen von Texten ein. Dieses können Sie über die Eigenschaft `TextAlignment` so einstellen, dass der Text linksbündig (`TextAlignment.Left`, Voreinstellung), rechtsbündig (`TextAlignment.Right`), zentriert (`TextAlignment.Center`) oder im Blocksatz (`TextAlignment.Justify`) erscheint.

Für die Ausrichtung des Textes ist das Rechteck wichtig, in dem der Text ausgegeben wird. Die X- und Y-Position wird durch den Punkt definiert, den Sie beim Zeichnen spezifizieren. Die Breite und Höhe des virtuellen Rechtecks ist zunächst auf unendlich eingestellt. Ein Text wird also nach rechts bzw. (bei rechtsbündigen Texten) nach links auch über das Fenster hinaus gezeichnet. Sie können die Breite und Höhe des Rechtecks aber über die Eigenschaften `MaxTextWidth` und `MaxTextHeight` bestimmen. Der Wert 0 (Voreinstellung) steht für eine unendliche Breite bzw. Höhe. Wenn der Text nicht mehr in das so definierte Rechteck passt, bestimmt übrigens die Eigenschaft `Trimming`, wie der Text abgeschnitten wird. Die Voreinstellung `TextTrimming.WordEllipsis` bewirkt, dass am Ende des noch passenden Textes an einer Wortgrenze drei Punkte gezeichnet werden.

Listing 698 zeigt nun, wie Sie in einem WPF-Fenster Text normal, zentriert und rechtsbündig ausgeben. Der wesentliche Trick dabei ist die Berechnung der inneren Fensterbreite und -höhe, für die ich keine andere Lösung gefunden habe, als die Breite des Fenster-Randes und bei der Höhe zusätzlich noch die Höhe des Fenster-Titels von der Gesamtbreite bzw. -höhe des Fensters abzuziehen.

Das Beispiel läuft in einem WPF-Fenster, das ein privates Feld `drawingVisual` vom Typ `DrawingVisual` enthält (und das entsprechend zum Zeichnen eingerichtet ist, siehe in Listing 688 in Rezept 343).

```
// Den DrawingContext für den DrawingVisual holen
using (DrawingContext dc = this.drawingVisual.RenderOpen())
{
   // Text normal ausgeben
   const int xMargin = 10;
   FormattedText formattedText = new FormattedText("Beispiel-Text, " +
      "der normal ausgegeben wird", Thread.CurrentThread.CurrentCulture,
      FlowDirection.LeftToRight, new Typeface("Tahoma"), 12,
      Brushes.Black);
   formattedText.MaxTextWidth = this.Width -
      SystemParameters.ThickVerticalBorderWidth - (xMargin * 2);
   dc.DrawText(formattedText, new Point(xMargin, 10));

   // Text zentriert und mittig ausgeben
   formattedText = new FormattedText("Beispiel-Text," +
      Environment.NewLine + "der auf dem Formular horizontal und " +
      "vertikal zentriert ausgegeben wird",
      Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
      new Typeface("Tahoma"), 12, Brushes.Black);
   formattedText.TextAlignment = TextAlignment.Center;
   formattedText.MaxTextWidth = this.Width -
      SystemParameters.ThickVerticalBorderWidth - (xMargin * 2);
   double x = (this.Width - SystemParameters.FixedFrameVerticalBorderWidth -
      formattedText.Width) / 2;
   double y = (this.Height - SystemParameters.WindowCaptionHeight -
      SystemParameters.FixedFrameHorizontalBorderHeight -
      formattedText.Height) / 2;
   dc.DrawText(formattedText, new Point(x, y));

   // Text rechtsbündig ausgeben
   formattedText = new FormattedText("Beispiel-Text," +
      Environment.NewLine + "der an einer definierten Y-Position " +
      "auf dem Fenster rechtsbündig " + Environment.NewLine +
      "mit einem kleinen Rand ausgegeben wird",
      Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
      new Typeface("Tahoma"), 12, Brushes.Black);
   formattedText.TextAlignment = TextAlignment.Right;
   formattedText.MaxTextWidth = this.Width -
      SystemParameters.ThickVerticalBorderWidth - (xMargin * 2);
   x = 10;
   y = this.Height - SystemParameters.WindowCaptionHeight -
      SystemParameters.ThickHorizontalBorderHeight - 55;
   dc.DrawText(formattedText, new Point(x, y));
}
```

Listing 698: Normales, rechtsbündiges und zentriertes Zeichnen eines Textes in einem WPF-Fenster

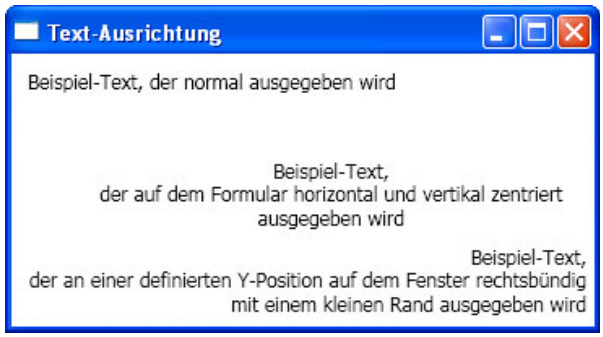


Abbildung 196: Das Beispiel in Aktion

Windows.Forms

Wenn Sie in einer `Windows.Forms`-Anwendung mithilfe der `DrawString`-Methode eines `Graphics`-Objekts Texte ausgeben, ist die Ausrichtung normalerweise (bzw. in unseren Breiten) linksbündig. Sie können aber auch recht einfach zentriert oder rechtsbündig zeichnen. Dazu übergeben Sie einer Variante der `DrawString`-Methode am letzten Argument ein `StringFormat`-Objekt, das das Format des auszugebenden Strings bestimmt. Idealerweise setzen Sie das Objekt ein, das die Eigenschaft `StringFormat.GenericTypographic` liefert, denn dieses ist bereits so eingestellt, dass die Breite eines Textes typografisch korrekt berechnet wird.

Hinweis

Bei der Verwendung der Eigenschaft `GenericTypographic` der `StringFormat`-Klasse sollten Sie beachten, dass diese Eigenschaft (die intern eine API-Funktion zur Ermittlung des Objekts aufruft) nicht immer eine neue Instanz eines `StringFormat`-Objekts erzeugt. Wenn Sie die Eigenschaften dieses Objekts verändern, sind Ihre Veränderungen unter Umständen auch bei späteren Zugriffen auf die `GenericTypographic`-Eigenschaft sichtbar. Der Beispielcode in Listing 699 (bei dem dieses Problem bereits auftrat) erzeugt deswegen jeweils ein neues `StringFormat`-Objekt, dessen Konstruktor allerdings die `GenericTypographic`-Eigenschaft der `StringFormat`-Klasse übergeben wird. Damit erzeugt das Beispiel eine Kopie dieses Objekts, die gefahrlos verändert werden kann.

Die Ausrichtung des Textes können Sie über die Eigenschaften `Alignment` (horizontale Ausrichtung) und `LineAlignment` (vertikale Ausrichtung) bestimmen. `StringAlignment.Near` gibt in der westlichen Welt eine linksbündige bzw. obere Ausrichtung an (*Near* bedeutet »nahe der Position, ab der gelesen wird«). `StringAlignment.Center` steht für eine zentrierte und `StringAlignment.Far` für eine rechtsbündige bzw. untere Ausrichtung. Diese Angaben beziehen sich immer auf ein Layout-Rechteck. Es ist also wichtig, die `DrawString`-Variante zu verwenden, der ein solches übergeben werden kann. Dieses Rechteck definieren Sie zumindest an der Position und mit der Größe der Fläche, auf der der Text ausgegeben werden soll. Wenn Sie einen Rand wünschen, können Sie diesen einfach der Position hinzuaddieren und (mit 2 multipliziert) von der Breite und der Höhe abziehen.

Listing 699 zeigt, wie dies prinzipiell programmiert wird. Das Beispiel gibt einen Text zunächst normal auf einem Formular aus. Ein weiterer Text wird dann horizontal und vertikal zentriert und ein dritter an einer festgelegten Y-Position rechtsbündig ausgegeben.

Zum Programmieren dieses Beispiels benötigen Sie ein Formular mit Namen *StartForm*, in dem Sie das `Paint`-Ereignis zuweisen, und Sie müssen die Namensräume `System`, `System.Drawing` und `System.Text` importieren.

```
private void StartForm_Paint(object sender, PaintEventArgs e)
{
   // Text normal ausgeben
   string text = "Beispiel-Text, der normal ausgegeben wird";
   using (Brush brush = new SolidBrush(Color.Black))
   {
      e.Graphics.DrawString(text, this.Font, brush, new RectangleF(10, 10,
         this.ClientRectangle.Width - 20, this.ClientRectangle.Height - 20));
   }

   // Text zentriert in einem Layout-Rechteck ausgeben
   text = "Beispiel-Text," + Environment.NewLine + "der auf dem Formular " +
      "horizontal und vertikal zentriert ausgegeben wird";
   StringFormat stringFormat =
      new StringFormat(StringFormat.GenericTypographic);
   stringFormat.Alignment = StringAlignment.Center;
   stringFormat.LineAlignment = StringAlignment.Center;
   using (Brush brush = new SolidBrush(Color.Black))
   {
      e.Graphics.DrawString(text, this.Font, brush, new RectangleF(0, 0,
         this.ClientRectangle.Width, this.ClientRectangle.Height),
         stringFormat);
   }

   // Text rechtsbündig in einem Layout-Rechteck ausgeben
   text = "Beispiel-Text," + Environment.NewLine + "der an einer " +
      "definierten Y-Position auf dem Formular rechtsbündig " +
       Environment.NewLine + "mit einem kleinen Rand ausgegeben wird";
   stringFormat = new StringFormat(StringFormat.GenericTypographic);
   stringFormat.Alignment = StringAlignment.Far;
   using (Brush brush = new SolidBrush(Color.Black))
   {
      e.Graphics.DrawString(text, this.Font, brush, new RectangleF(
         10, this.ClientRectangle.Height - 45,
         this.ClientRectangle.Width - 20, 45), stringFormat);
   }
}
```

Listing 699: Zentriertes und rechtsbündiges Zeichnen von Strings in einem Windows.Forms-Formular

349 Strings beim Zeichnen wortgerecht umbrechen

Beim Drucken oder beim Entwickeln von Steuerelementen, die Texte ausgeben, ist es häufig notwendig, auszugebende Strings an einem rechten Rand zu umbrechen. Dabei soll natürlich nicht mitten in einem Wort, sondern vor Wörtern, die den rechten Rand überschreiten würden, umbrochen werden. Dieses Problem können Sie sehr einfach lösen.

WPF

In WPF-Anwendungen definieren Sie ein Rechteck, in dem Text ausgegeben werden soll, über die X- und Y-Position, an der Sie den Text ausgeben, und über die Eigenschaften `MaxTextWidth` und `MaxTextHeight` des `FormattedText`-Objekts, das Sie zum Zeichnen verwenden. Diese Eigenschaften sind per Voreinstellung mit 0 definiert, was für eine unendliche Breite bzw. Höhe steht. Die Eigenschaft `Trimming` bestimmt, wie der Text abgeschnitten wird, wenn er nicht komplett in das Rechteck passt.

- `TextTrimming.WordEllipsis` (Voreinstellung): Nicht mehr passender Text wird an einer Wortgrenze abgeschnitten (sodass also keine Wörter mitten im Wort abgeschnitten werden) und durch drei Punkte ergänzt.
- `TextTrimming.CharacterEllipsis`: Nicht mehr passender Text wird an einem Zeichen abgeschnitten (also auch mitten im Wort) und durch drei Punkte ergänzt
- `TextTrimming.None`: Text wird nicht abgeschnitten.

Das folgende Beispiel demonstriert den Umgang mit diesen Eigenschaften an einem längeren Text, der mit etwas Rand in einem Fenster ausgegeben wird. Das Beispiel erfordert ein WPF-Fenster, das ein privates Feld `drawingVisual` vom Typ `DrawingVisual` enthält (und das entsprechend zum Zeichnen eingerichtet ist, siehe in Listing 688 in Rezept 343).

```
// Den DrawingContext für den DrawingVisual holen
using (DrawingContext dc = this.drawingVisual.RenderOpen())
{
   // Die Position und die Maße des Text-Rechtecks bestimmen
   const double x = 30;
   const double y = 30;
   double textRectangleWidth = this.Width -
      SystemParameters.FixedFrameVerticalBorderWidth - (2 * x);
   double textRectangleHeight = this.Height -
      SystemParameters.WindowCaptionHeight -
      SystemParameters.FixedFrameHorizontalBorderHeight - (2 * y);

   // FormattedText-Instanz für den Text erzeugen
   FormattedText formattedText = new FormattedText("Weit draußen in den " +
      "unerforschten Einöden eines total aus der Mode gekommenen " +
```

Listing 700: Zeichnen eines Textes mit wortgerechtem Umbruch in WPF

```
        "Ausläufers des westlichen Spiralarms der Galaxis leuchtet " +
        "unbeachtet eine kleine gelbe Sonne. Um sie kreist in einer " +
        "Entfernung von ungefähr achtundneunzig Millionen Meilen ein " +
        "absolut unbedeutender, kleiner blaugrüner Planet, dessen vom " +
        "Affen stammende Bioformen so erstaunlich primitiv sind, dass " +
        "sie Digitaluhren noch immer für eine unwahrscheinlich tolle " +
        "Erfindung halten.",
        Thread.CurrentThread.CurrentCulture, FlowDirection.LeftToRight,
        new Typeface("Tahoma"), 14, Brushes.Black);

    // Die Maximalbreite und -höhe definieren
    formattedText.MaxTextWidth = textRectangleWidth;
    formattedText.MaxTextHeight = textRectangleHeight;

    // Die Art des Abschneidens (auf den Defaultwert) festlegen
    formattedText.Trimming = TextTrimming.WordEllipsis;

    // Den Text zeichnen
    dc.DrawText(formattedText, new Point(x, y));
}
```

Listing 700: Zeichnen eines Textes mit wortgerechtem Umbruch in WPF (Forts.)

Abbildung 197 zeigt das Beispiel zu diesem Rezept in Aktion.

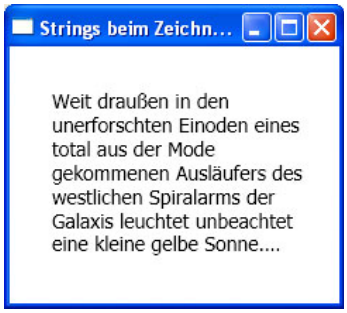


Abbildung 197: In einem virtuellen Layout-Rechteck ausgegebener Text wird automatisch wortgerecht umbrochen und mit Auslassungszeichen versehen, wenn er zu lang ist

Windows.Forms

In einer `Windows.Forms`-Anwendung können Sie Texte in einem Layout-Rechteck ausgeben, wie ich es in Rezept 348 zeige. Ein solches sorgt automatisch dafür, dass Text wortgerecht umbrochen wird, wie es Abbildung 198 (das Beispiel aus Rezept 348 mit einem verkleinerten Formular) zeigt.

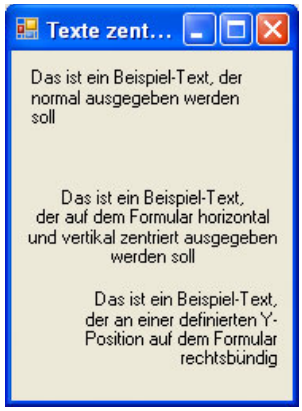


Abbildung 198: In einem Layout-Rechteck ausgegebener Text wird automatisch wortgerecht umbrochen

Ähnlich wie bei WPF können Sie eine Trimmung zu langer Texte erreichen, indem Sie am letzten Argument der `DrawString`-Methode des `Graphics`-Objekts ein `StringFormat`-Objekt übergeben, dessen `Trimming`-Eigenschaft Sie entsprechend eingestellt haben.

Wollen Sie die resultierenden Zeilen eines umbrochenen Textes jedoch Zeile für Zeile einzeln ausgeben (z. B. weil Sie den Zeilenabstand selbst definieren wollen), können Sie mit diesem automatischen Umbruch nicht viel anfangen. Dann benötigen Sie eine Methode, die Ihnen alle Zeilen eines Textes wortgerecht umbrochen liefert.

Hinweis

In WPF können Sie den Zeilenabstand über die Eigenschaft `LineHeight` des `FormattedText`-Objekts, das Sie zum Zeichnen verwenden, bestimmen. Deswegen verzichte ich auf eine WPF-Version der `WordWrapString`-Methode.

Diese Arbeit übernimmt die Methode `WordWrapString` in Listing 701. Diese Methode arbeitet nach demselben Prinzip wie die gleichnamige Methode aus dem Rezept 29. Die Grundidee dabei ist, die einzelnen Zeilen des übergebenen Strings durchzugehen und diese in einzelne Wort-Token zu zerlegen.

Ein Wort-Token ist dabei eine Folge von Wort-Zeichen, die von einer beliebigen Anzahl an Nicht-Wort-Zeichen oder Bindestrichen abgeschlossen werden. Die grundsätzliche Arbeitsweise der Methode habe ich bereits in Rezept 29 beschrieben, weswegen ich hier darauf verzichte. Einer der Unterschiede ist, dass nun neben dem Text und der maximalen Breite ein `Graphics`-Objekt und die zu verwendende Schriftart übergeben werden. Die Breite bezieht sich in dieser `WordWrapString`-Variante auch nicht auf die Anzahl der Zeichen, sondern auf die tatsächliche Breite in Pixeln. Ein weiterer Unterschied ist, dass die Methode eine `StringCollection` mit den ermittelten Zahlen zurückgibt. Schließlich wird die Breite des jeweiligen Token nicht nach seiner Zeichen-Anzahl, sondern über die `MeasureString`-Methode des übergebenen Graphics-Objekts berechnet.

Etwas eigenartig war, dass das `StringFormat`-Objekt, das die `GenericTypographic`-Eigenschaft zurückliefert, in meinen Tests dazu führte, dass die gemessene Breite etwas zu klein war, weswegen der Text am rechten Rand nicht immer korrekt umbrach. Deswegen habe ich das Objekt verwendet, das die `GenericDefault`-Eigenschaft zurückgibt.

Zum Kompilieren dieser Methode müssen Sie die Namensräume `System`, `System.Collections.Generic`, `System.Drawing` und `System.Text.RegularExpressions` importieren.

```
public static List<String> WordWrapString(string text,
   int maxWidth, Graphics g, Font font)
{
   // List-Instanz für den umbrochenen Text erzeugen
   List<String> wrappedText = new List<String>();

   // StringFormat-Objekt für die Ermittlung der Breite
   // der einzelnen Wort-Token ermitteln
   StringFormat stringFormat = StringFormat.GenericDefault;

   if (text != null && text.Length > 0)
   {
      // Tabs durch Leerzeichen ersetzen um
      // Darstellungs-Probleme zu vermeiden
      text = text.Replace("\t", "   ");

      // Aufteilen des Textes in einzelne Zeilen
      string[] rows = Regex.Split(text, Environment.NewLine);
      foreach (string row in rows)
      {
         // Den Sonderfall einer Leerzeile abfangen)
         if (row == null || row.Length == 0)
         {
            wrappedText.Add(String.Empty);
         }
         else
         {
            // Text in einzelne Wort-Token aufsplitten
            // Als Trennzeichen werden alle Nicht-Wort-Zeichen und
            // der Bindestrich verwendet
            MatchCollection matches = Regex.Matches(row, @"\w{1,}(\W|-){0,}");

            // Die einzelnen Wort-Token durchgehen
            string currentRow = null;
            int currentWidth = 0;
```

Listing 701: Methode zum wortgerechten Umbrechen von Strings zur Ausgabe auf einem Graphics-Objekt

```
for (int i = 0; i < matches.Count; i++)
{
   Match match = matches[i];

   // Den Sonderfall behandeln, dass das erste Wort-Token
   // zu lang ist
   int tokenLength = (int)g.MeasureString(match.Value,
     font, 0, stringFormat).Width;
   if (i == 0 && currentWidth + tokenLength > maxWidth)
   {
      // Aktuelles Token ablegen
      wrappedText.Add(match.Value);
      currentRow = null;
      currentWidth = 0;
   }
   else
   {
      if (currentWidth + tokenLength <= maxWidth)
      {
         // Aktuelles Token ablegen
         currentRow += match.Value;
      }
      else
      {
         // Das Wort passt nicht mehr:
         // Zeile ablegen
         wrappedText.Add(currentRow);

         // Token ablegen
         currentRow = match.Value;

         // Die aktuelle Breite zurücksetzen
         currentWidth = 0;
      }

      // Die aktuelle Breite um die Wortlänge hochzählen
      currentWidth += tokenLength;
   }
}

// Eine eventuelle Rest-Zeile ablegen
if (currentRow != null)
{
   wrappedText.Add(currentRow);
}
```

Listing 701: Methode zum wortgerechten Umbrechen von Strings zur Ausgabe auf einem Graphics-Objekt (Forts.)

```
            }
        }
    }
    // Ergebnis zurückgeben
    return wrappedText;
}
```

Listing 701: Methode zum wortgerechten Umbrechen von Strings zur Ausgabe auf einem Graphics-Objekt (Forts.)

Listing 702 zeigt eine beispielhafte Anwendung im `Paint`-Ereignis eines Formulars. Diese Methode entscheidet anhand des Zustands zweier `OptionButtons`, ob über ein Layout-Rechteck oder über `WordWrapText` umbrochen werden soll. Beim Umbruch über `WordWrapText` wird ein im Formular über eine `TextBox` gesetzter Zeilenabstand berücksichtigt.

```
private void StartForm_Paint(object sender, PaintEventArgs e)
{
    string text = "Dieser Test-Text enthält einige spezielle " +
        "Trenn~Zeichen und auch schon einmal mehrere    Leer-Zeichen " +
        "und mehrere hintereinander liegende Binde----Striche.\r\n" +
        "Außerdem sind Tabs:\t enthalten \r\n" +
        "\r\n" +
        "DasIstEinWortDasZuLangIst\r\n" +
        "\r\n" +
        "Test-Wörter mit []: Test[Wort1] Test[Wort2]\r\n" +
        "\r\n" +
        "Test-Wörter mit + getrennt: Testwort1+Testwort2\r\n" +
        "\r\n" +
        "Test-Wörter mit / getrennt: Testwort1/Testwort2\r\n" +
        "\r\n" +
        "Test-Wörter mit \\ getrennt: Testwort1\\Testwort2" +
        "\r\n\r\nAktuelle Breite: " + this.ClientRectangle.Width;

    if (this.wrapViaLayoutRectangleRadioButton.Checked)
    {
        using (Brush brush = new SolidBrush(Color.Black))
        {
        e.Graphics.DrawString(text, this.Font, brush,
            new RectangleF(10, 10, this.ClientRectangle.Width - 20,
            this.ClientRectangle.Height - 20));
        }
    }
    else
    {
```

Listing 702: Beispielhaftes Zeichnen eines Textes mit wortgerechten Umbrüchen

```
      int xMargin = 10;
      int y = 10;

      // Zeilenhöhe über die Höhe eines Zeichens ermitteln
      float rowHeight = e.Graphics.MeasureString("X", this.Font).Height;
      // Zeilenhöhe etwas erhöhen
      try
      {
         rowHeight *= Convert.ToSingle(this.lineSpacingTextBox.Text);
      }
      catch { }

      // Die umbrochenen Zeilen ermitteln
      int width = this.ClientRectangle.Width - (xMargin * 2);
      StringCollection wrappedText = PrintUtils.WordWrapString(text, width,
         e.Graphics, this.Font);

      // Zeichnen
      using (Brush brush = new SolidBrush(Color.Black))
      {
         foreach (string row in wrappedText)
         {
            e.Graphics.DrawString(row, this.Font, brush,
               xMargin, y);
            y += (int)rowHeight;
         }
      }
   }
}
```

Listing 702: Beispielhaftes Zeichnen eines Textes mit wortgerechten Umbrüchen (Forts.)

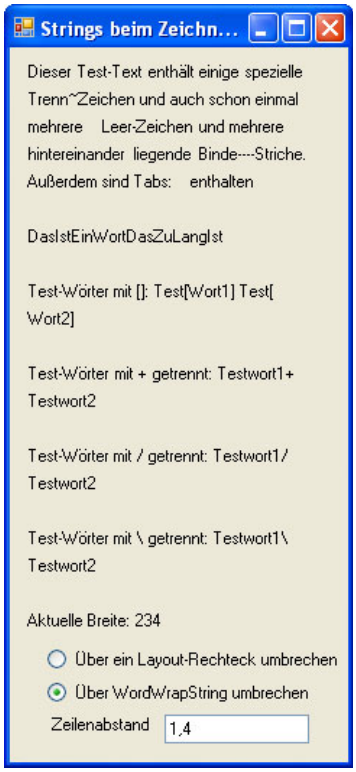


Abbildung 199: Das Beispiel in Aktion. Umbrochen wird gerade über WordWrapString

COM-Interop mit Office

350 Laufende COM-Komponenten-Instanzen referenzieren

Bei der Verwendung von COM-Komponenten, die gleichzeitig interaktiv verwendbare Anwendungen sind (wie es z. B. bei den Microsoft-Office-Anwendungen der Fall ist), ist es häufig sinnvoll, eventuell bereits ausgeführte Instanzen der Anwendung für Fernsteuerungszwecke zu referenzieren. Die Verwendung einer bereits laufenden Instanz führt dazu, dass das Programm schneller ausgeführt wird, weniger Ressourcen verbraucht und die erzeugte Instanz nicht wieder geschlossen werden muss. Das ist z. B. dann der Fall, wenn Sie über Outlook eine E-Mail versenden wollen und davon ausgehen, dass die meisten Anwender Outlook permanent geöffnet haben.

Um eine laufende Instanz einer COM-Komponente zu referenzieren, verwenden Sie die Methode `GetActiveObject` der Klasse `Marshal` aus dem Namensraum `System.Runtime.InteropServices`. Dieser Methode übergeben Sie die Prog-Id der COM-Klasse (siehe »COM-Komponenten mit später Bindung verwenden« in der Einführung auf der Buch-CD). `GetActiveObject` gibt eine `object`-Referenz auf die Instanz zurück, falls gerade eine ausgeführt wird. Wird keine Instanz der COM-Klasse ausgeführt, erzeugt diese Methode eine Ausnahme vom Typ `COMException` mit der Meldung »Vorgang nicht verfügbar«.

Listing 703 setzt dieses Wissen um. Das Programm versucht zunächst, eine laufende Outlook-Instanz zu referenzieren. Der Aufruf der `GetActiveObject`-Methode erfolgt in einer Ausnahmebehandlung (die die Ausnahme allerdings verwirft) um darauf zu reagieren, dass Outlook gerade nicht ausgeführt wird. Da Outlook auf jeden Fall verwendet werden soll, fragt das Programm danach über einen Vergleich der erhaltenen `object`-Referenz auf `null` ab, ob eine Instanz referenziert werden konnte. Ist dies der Fall, wird die `object`-Referenz in den Datentyp `Outlook.Application` konvertiert. Das Programm arbeitet mit früher Bindung, benötigt also Wrapper-Assemblies für die Outlook-COM-Komponenten (siehe »COM-Komponenten mit früher Bindung verwenden« in der Einführung auf der Buch-CD). Sie können aber natürlich auch mit später Bindung arbeiten (was ich aber nicht im Listing darstellen wollte).

Kann keine Outlook-Instanz referenziert werden, erzeugt das Programm einfach eine neue, wobei die Variable `outlookInstanceCreated` für das spätere Beenden von Outlook auf `true` gesetzt wird. Danach erzeugt Listing 703 eine neue E-Mail und versendet diese. Wurde Outlook neu gestartet, beendet das Programm diese Instanz schließlich.

Zum Kompilieren dieses Programms sollten Sie eine Referenz auf die mit Visual Studio 2008 mitgelieferte Outlook-Interop-Assembly *Microsoft.Office.Interop.Outlook* erstellen. Eine Referenzierung der Version 12 funktioniert auch für Outlook 2003. Falls Sie Visual Studio 2008 nicht besitzen, sollten Sie die »Microsoft Office Primary Interop Assemblies« (PIAs) von Microsoft herunterladen und installieren (suchen Sie bei *www.microsoft.com/downloads* nach »Primary Interop Assemblies«). Die *Microsoft.Office.Interop.Outlook* ist Bestandteil der PIAs. Außerdem müssen Sie die Namensräume `System` und `System.Runtime.InteropServices` importieren.

```
// Versuch, eine eventuell bereits ausgeführte
// Outlook-Instanz zu referenzieren
object outlookObject = null;
try
{
   outlookObject = Marshal.GetActiveObject("Outlook.Application");
}
catch {}

Outlook.Application outlook = null;
bool outlookInstanceCreated = false;
if (outlookObject != null)
{
   // Wenn eine Instanz referenziert werden konnte, wird diese in den
   // passenden .NET-Typ konvertiert (frühe Bindung)
   outlook = (Outlook.Application)outlookObject;
}
else
{
   // Wenn keine Instanz referenziert werden konnte,
   // wird einfach eine neue erzeugt
   outlookInstanceCreated = true; // Merker für das Schließen unten
   outlook = new Outlook.ApplicationClass();
}

// Neue E-Mail erzeugen und versenden
Outlook.MailItem mailItem = (Outlook.MailItem)outlook.CreateItem
   (Outlook.OlItemType.olMailItem);
mailItem.To = "leser@home.de";
mailItem.Subject = "Codebook";
mailItem.Body = "Hallo Leser, " + Environment.NewLine +
  "wie gefällt dir mein Codebook?";
mailItem.Send();

// Outlook beenden, wenn eine neue Instanz erzeugt wurde
if (outlookInstanceCreated)
{
   outlook.Quit();
}
```

Listing 703: Versenden einer E-Mail über eine ausgeführte bzw. eine neue Outlook-Instanz

Hinweis

Das Versenden von E-Mails über Outlook wird in Rezept 212 grundlegend behandelt. Dort finden Sie den folgenden Hinweis noch einmal, den ich hier nur für den Fall anbringe, dass Sie sich über Outlook wundern ...

Outlook zeigt ab der Version 2000, Service Pack 2 beim Versuch, über externe Programme E-Mails zu versenden, aus Sicherheitsgründen eine Warnmeldung an (Abbildung 200). Der Ja-Schalter dieses Dialogs kann für fünf Sekunden nicht betätigt werden. Damit verhindert Microsoft, dass Viren-Programme unkontrolliert E-Mails versenden (allerdings auf Kosten der Benutzbarkeit ...). Dieses in der Regel eher lästige Feature kann nicht einfach abgeschaltet werden. Sie können jedoch an der Adresse *www.mapilab.com/de/outlook/security* das Tool *Advanced Security for Outlook* downloaden, das Ihnen diese lästige Bestätigung optional abnimmt. Beim externen Zugriff auf Outlook zeigt Advanced Security for Outlook einen Dialog an, in dem Sie entscheiden können, ob Sie den Zugriff erlauben wollen oder nicht. Da Sie auch einstellen können, dass diese Wahl für alle Zugriffe der entsprechenden Anwendung gilt, ersparen Sie (bzw. der Anwender, der Ihre Programme einsetzt) sich ab dem nächsten Zugriff die lästige Outlook-Warnmeldung.

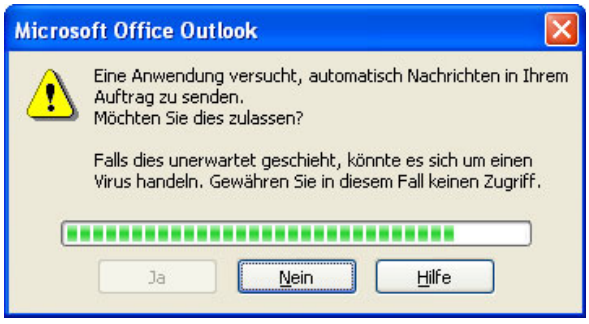


Abbildung 200: Warnmeldung ab Outlook 2000 SP 2 beim Versenden einer E-Mail über ein externes Programm

351 Word-Dokumente öffnen

Word-Dokumente können Sie in .NET nicht direkt öffnen und anzeigen.

Tipp

Sie können in WPF-Anwendungen allerdings den Umweg über ein XPS-Dokument (XML Paper Specification Dokument) gehen, das sowohl in Word als auch in einer WPF-Anwendung bearbeitet werden kann. Für Word 2007 benötigen Sie dazu ein Add-In, das Sie bei Microsoft downloaden können. Suchen Sie dazu bei *www.microsoft.com/downloads* nach »Microsoft Office Add-in: Microsoft Save as XPS«. In einer WPF-Anwendung können Sie ein XPS-Dokument über eine `FixedDocument`-Instanz bearbeiten und über ein `DocumentViewer`-Steuerelement anzeigen.

Zur Bearbeitung von Word-Dokumenten müssen Sie Word fernsteuern, also eine neue Word-Instanz erzeugen (oder eine vorhandene referenzieren, siehe Rezept 350) und deren Eigenschaften und Methoden einsetzen. Idealerweise setzen Sie dazu die frühe Bindung ein (siehe »COM-Komponenten mit später Bindung verwenden« in der Einführung auf der Buch-CD). Dazu referenzieren Sie die zu den »Primary Interop Assemblies« (siehe Rezept 350) gehörende Assembly *Microsoft.Office.Interop.Word*.

Die Fernsteuerung von Word ist relativ einfach. Word stellt, wie andere Office-Anwendungen auch, in seiner Klassenbibliothek ein Objektmodell zur Verfügung, das eine Word-Instanz und deren Dokumente repräsentiert. Das einzig Problematische ist vielleicht, dass Sie dieses Objektmodell nicht kennen. Die Dokumentation finden Sie für Word 2003 (ehrlich gesagt mag ich Office 2007 nicht ...) an der Adresse *msdn.microsoft.com/en-us/library/aa272713(office.11).aspx* oder indem Sie bei *msdn.microsoft.com* nach »Word object model« suchen.

Zum Kompilieren des folgenden Beispiels müssen Sie PIA-Assembly *Microsoft.Office.Interop.Word* referenzieren und die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

Zur Fernsteuerung benötigen Sie zunächst eine Instanz der Klasse `Word.ApplicationClass` (die die Schnittstelle `Word.Application` implementiert). Wenn Sie diese nicht aus einer laufenden Instanz referenzieren (siehe Rezept 350), sollten Sie darauf achten, dass eine neue Word-Instanz immer zunächst unsichtbar ausgeführt wird. Wenn Sie diese sichtbar schalten wollen, müssen Sie deren `Visible`-Eigenschaft auf `true` setzen:

```
// Word-Instanz erzeugen und sichtbar schalten
Microsoft.Office.Interop.Word.Application word =
   new Microsoft.Office.Interop.Word.ApplicationClass();
word.Visible = true;
```

Listing 704: Erzeugen einer Word-Instanz und Sichtbar-Schalten derselben

Sie können die Instanz aber auch unsichtbar verwenden, wenn der Anwender nicht sehen soll, was im Hintergrund passiert. Da der Bildschirm nicht aktualisiert werden muss, ist die Verwendung einer unsichtbaren Instanz auch etwas performanter. Sie müssen aber gerade bei unsichtbaren Instanzen darauf achten, diese (über die `Quit`-Methode) schließlich wieder zu beenden, da der Prozess ansonsten weiter ausgeführt wird und Ressourcen verbraucht.

Um ein Dokument zu öffnen, rufen Sie nun die `Open`-Methode der `Documents`-Auflistung auf. Die `Documents`-Auflistung verwaltet alle Dokumente, die in der Word-Instanz geöffnet sind. Im Prinzip handelt es sich dabei um eine ganz normale Auflistung. Der wesentliche Unterschied zu .NET-Auflistungen ist, dass COM-Auflistungen (und speziell `Documents`) in der Regel 1-basiert sind (also nicht bei 0 beginnen).

`Open` ist eine typische (Office-)COM-Methode mit einer Vielzahl an optionalen Argumenten. Die meisten dieser Argumente sind für die Praxis eher unwichtig, hier können Sie also einfach `System.Reflection.Missing.Value` übergeben. Da die Argumente der `Open`-Methode als `object`

und By-Reference deklariert sind, können Sie diesen Wert jedoch nicht direkt einsetzen, sondern müssen dazu eine `object`-Variable verwenden, wie es Listing 705 zeigt. In diesem Beispiel wird das Dokument *hitchhiker.doc* im Ordner der Anwendung geöffnet.

```
// Variable für den Wert Missing.Value für nicht belegte optionale Argumente
object missing = Missing.Value;

// Dokument öffnen
string appPath = System.IO.Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
object fileName = Path.Combine(appPath, "Hitchhiker.doc");
word.Documents.Open(ref fileName, ref missing, ref missing, ref missing,
   ref missing, ref missing, ref missing, ref missing, ref missing,
   ref missing, ref missing, ref missing, ref missing, ref missing,
   ref missing, ref missing);
```

Listing 705: Einfaches Öffnen eines Word-Dokuments

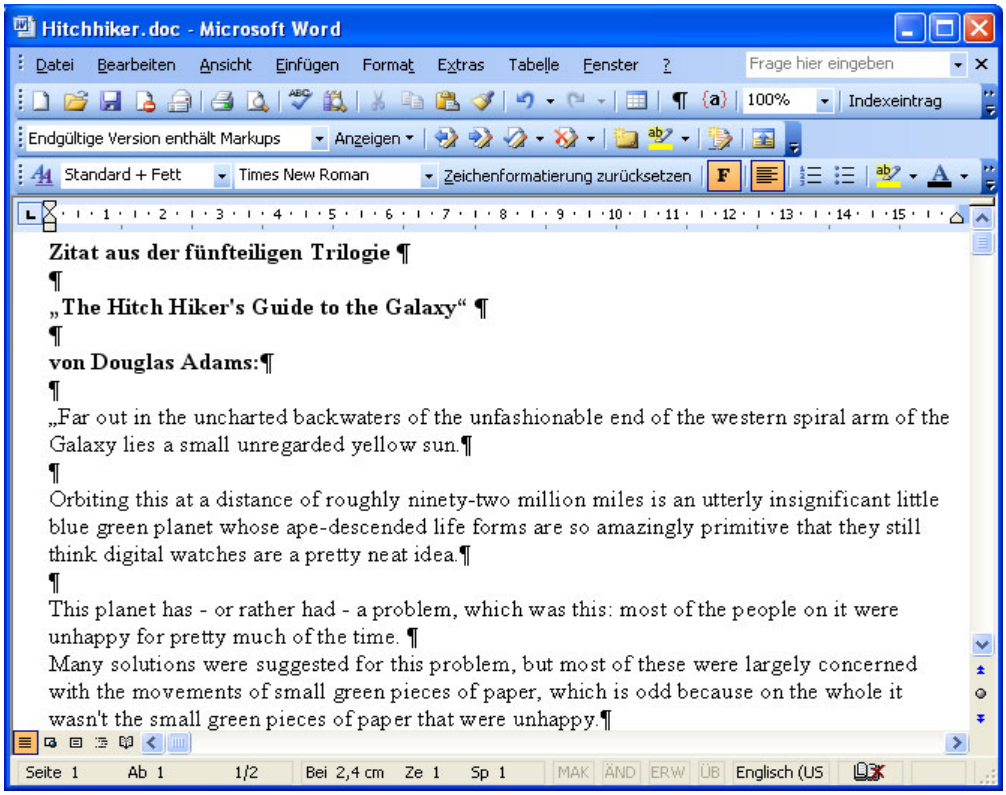


Abbildung 201: Das Beispielprogramm hat Word gestartet und das Dokument geöffnet

Hinweis

Ein guter »Trick« zur Fernsteuerung von Office-Anwendung ist, dass Sie in der Office-Anwendung (sofern möglich) einfach ein Makro aufzeichnen, das die gewünschten Aktionen ausführt. Dann wissen Sie schon einmal, welche Eigenschaften und Methoden Sie einsetzen müssen. Das einzige Problem dabei ist, dass der Office-Makrorekorder (leider) eine spezielle Visual-Basic-Syntax einsetzt, bei der die Argumente einer Methode unter der Angabe des Argumentnamens verwendet werden. Ein Makro, das das aktuelle Dokument ausdruckt, sieht z. B. so aus:

```
Application.PrintOut FileName:="", Range:=wdPrintAllDocument, Item:= _
  wdPrintDocumentContent, Copies:=1, Pages:="", PageType:=wdPrintAllPages, _
  ManualDuplexPrint:=False, Collate:=True, Background:=True, PrintToFile:= _
  False, PrintZoomColumn:=0, PrintZoomRow:=0, PrintZoomPaperWidth:=0, _
  PrintZoomPaperHeight:=0
```

Listing 706: Ein Word-Makro, das das aktuelle Dokument ausdruckt

Wenn Sie diese Syntax nach C# umsetzen, müssen Sie die Argumente natürlich in Klammern setzen und die zu übergebenden Argumente an dem angegebenen Namen (vor dem :=) identifizieren und korrekt platzieren. Optionale und für die auszuführende Aktion unwichtige Argumente können Sie einfach durch `Missing.Value` ersetzen. Die Aufzählungen der verwendeten Konstanten (z. B. `wdPrintAllDocument`) können Sie in Visual Studio 2008 über den Objektbrowser ermitteln, indem Sie nach dem Konstantennamen suchen.

In C# würde dieser Methodenaufruf dann so aussehen:

```
object background = true;
object range = Microsoft.Office.Interop.Word
   .WdPrintOutRange.wdPrintAllDocument;
object item = Microsoft.Office.Interop.Word
   .WdPrintOutItem.wdPrintDocumentContent;
object copies = 1;
object pageType = Microsoft.Office.Interop.Word
   .WdPrintOutPages.wdPrintAllPages;
object printToFile = false;
object collate = true;
object manualDuplexPrint = false;
object printZoomColumn = 0;
object printZoomRow = 0;
object printZoomPaperWidth = 0;
object printZoomPaperHeight = 0;

word.PrintOut(ref background, ref missing, ref range, ref missing,
   ref missing, ref missing, ref item, ref copies, ref missing, ref pageType,
```

Listing 707: Ein nach C# umgesetzter Aufruf einer über den Office Makrorekorder erzeugten Visual Basic COM-Methode

```
    ref printToFile, ref collate, ref missing, ref missing,
    ref manualDuplexPrint, ref printZoomColumn, ref printZoomRow,
    ref printZoomPaperWidth, ref printZoomPaperHeight);
```

Listing 707: Ein nach C# umgesetzter Aufruf einer über den Office Makrorekorder erzeugten Visual Basic COM-Methode (Forts.)

352 Word-Dokumente basierend auf einer Dokumentenvorlage erzeugen, füllen, ausdrucken und speichern

In der Praxis kommt es häufig vor, dass eine Anwendung ein Word-Dokument erzeugen und mit Daten füllen muss, das auf einer Dokumentenvorlage basiert. Damit erleichtern Sie dem Anwender – der in der Regel mit Word sehr gut umgehen kann – die Anpassung auszudruckender Dokumente. Ich drucke z. B. meine Rechnungen in einem selbst geschriebenen Buchführungs-Programm über eine spezielle Word-Dokumentenvorlage aus.

Hinweis

Das in diesem Rezept behandelte Fernsteuern von Word ist nicht besonders schnell und außerdem problematisch, wenn das Ganze in einer Webanwendung ausgeführt wird. Die reduzierte Geschwindigkeit liegt daran, dass der Zugriff auf Word über Prozessgrenzen hinweg über COM erfolgt. Dieses Problem können Sie über Komponenten lösen, die Word-Dokumente bearbeiten können. Eine mögliche Komponente (die ich allerdings noch nicht ausprobiert habe) ist *Aspose.Words* (*www.aspose.com/Products/Aspose.Words*), die es erlaubt, Word-Dokumente direkt (also ohne Word) zu bearbeiten. Solche Komponenten lösen auch das Problem, das in Webanwendungen entsteht, wenn Word-Instanzen aufgrund von Fehlern in der Anwendung nicht geschlossen werden und folglich als Prozess im Speicher bleiben (und irgendwann den Webserver überlasten).

Als Beispiel für das Erstellen eines Word-Dokuments über das Fernsteuern von Word verwende ich eine einfache Dokumentenvorlage, die einen Brief darstellen soll. Diese Vorlage ist mit einigen Textmarken versehen (was schon der ganze Trick ist). In Abbildung 202 sind die Textmarken über spitze Klammern gekennzeichnet.

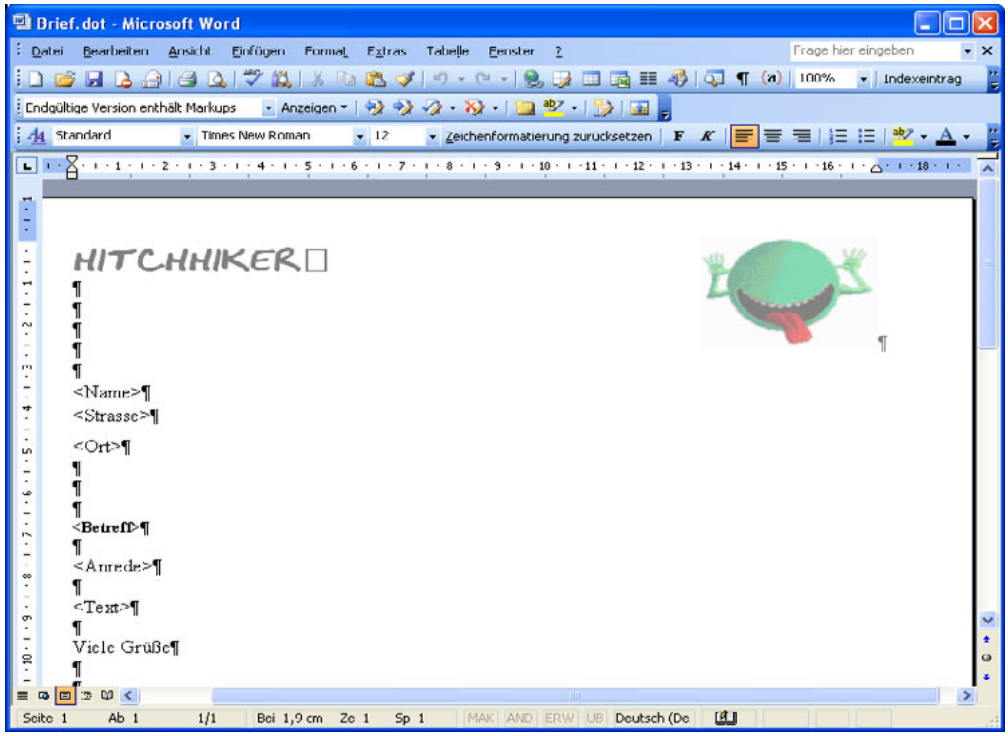


Abbildung 202: Eine einfache Dokumentenvorlage für einen Brief

Hinweis

Falls Sie nicht wissen, wie Sie in Word eine solche Dokumentenvorlage erzeugen: Erstellen Sie ein neues Dokument und fügen Sie den gewünschten Text und die gewünschten Grafiken ein. Erstellen Sie an den Stellen, an denen das Programm Texte einfügen soll, Textmarken, indem Sie die Stelle markieren und den Befehl EINFÜGEN / TEXTMARKE wählen. Speichern Sie das Dokument dann als Dokumentenvorlage (**.dot*) ab. Wenn Sie die Dokumentenvorlage nur für eine spezielle Anwendung benötigen, bietet es sich an, diese im Anwendungsordner zu speichern. Ansonsten können Sie auch den Ordner verwenden, der für Ihre Dokumentenvorlagen vorgesehen ist (per Voreinstellung ist das unter Windows XP der Ordner *C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Vorlagen*).

Um nun basierend auf einer Dokumentenvorlage ein neues Dokument zu erzeugen, erstellen Sie zunächst eine neue Word-Instanz (bzw. referenzieren eine vorhandene, wie ich es prinzipiell im Rezept 350 beschreibe) und rufen die `Add`-Methode der `Documents`-Auflistung auf. Grundlagen dazu finden Sie im Rezept 351.

Der `Add`-Methode können Sie am ersten Argument den Dateinamen einer Dokumentenvorlage übergeben. Listing 708 zeigt dies am Beispiel der Dokumentenvorlage *Brief.dot*, die im Ordner der Anwendung erwartet wird.

Zum Kompilieren des Programms müssen Sie die Namensräume `System`, `System.IO` und `System.Reflection` importieren.

```
// Word-Instanz erzeugen und sichtbar schalten
Microsoft.Office.Interop.Word.Application word =
   new Microsoft.Office.Interop.Word.ApplicationClass();
word.Visible = true;

// Neues Dokument basierend auf der Dokumentenvorlage Brief.dot erzeugen
object missing = Missing.Value;
string appPath = System.IO.Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
object template = Path.Combine(appPath, "Brief.dot");
word.Documents.Add(ref template, ref missing, ref missing,
   ref missing);
```

Listing 708: Erzeugen eines neuen Word-Dokuments auf der Basis einer Dokumentenvorlage

Nun können Sie die einzelnen Textmarken anspringen und den gewünschten Text einfügen. Dazu verwenden Sie die `GoTo`-Methode der `Selection`-Eigenschaft, der Sie am ersten Argument die Word-Konstante `WdGoToItem.wdGoToBookmark` übergeben (weil Sie eine Textmarke anspringen wollen). Da dieses Argument (wie auch die anderen) ein COM-Variant-Typ ist, der By-Reference übergeben wird, müssen Sie zur Übergabe eine `object`-Variable verwenden. Das zweite und dritte Argument sind für unseren Zweck unwichtig. Hier übergeben Sie den Wert `Missing.Value`, wieder als `object`-Variable. Am letzten Argument übergeben Sie den Namen der Textmarke.

```
object what = Microsoft.Office.Interop.Word.WdGoToItem.wdGoToBookmark;
object name = "Name";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Spongebob Squarepants");

name = "Strasse";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Ananas");

name = "Ort";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Bikini Bottom");

name = "Betreff";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Besuch");
```

Listing 709: Anspringen von Textmarken und Einfügen von Text

```
name = "Anrede";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("Hallo Spongebob,");

name = "Text";
word.Selection.GoTo(ref what, ref missing, ref missing, ref name);
word.Selection.TypeText("ich würde Dich gerne mal besuchen. Hast du Zeit?");
```

Listing 709: Anspringen von Textmarken und Einfügen von Text (Forts.)

Das Dokument können Sie über die `PrintOut`-Methode des aktiven Dokuments ausdrucken (wobei viele für die Praxis eher unwichtige optionale Argumente belegt werden müssen). Das erste Argument ist allerdings wichtig, wenn Sie Word direkt nach dem Drucken wieder schließen, denn es steuert das Drucken im Hintergrund. In der Voreinstellung wird im Hintergrund gedruckt, was bedeutet, dass der Druck abgebrochen wird, wenn Word direkt danach beendet wird. Deswegen übergibt das Beispiel hier eine `Object`-Variable mit dem Wert `false`. Über die `SaveAs`-Methode können Sie das Dokument dann speichern (wieder mit vielen optionalen Argumenten) und Word schließlich über die `Quit`-Methode der Word-Instanz beenden:

```
// Das Dokument (nicht im Hintergrund) ausdrucken ...
object falseValue = false;
word.ActiveDocument.PrintOut(ref falseValue, ref missing, ref missing,
   ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing);

// ... und speichern
object fileName = Path.Combine(Application.StartupPath, "Brief.doc");
word.ActiveDocument.SaveAs(ref fileName, ref missing, ref missing,
   ref missing, ref missing, ref missing, ref missing, ref missing,
   ref missing, ref missing, ref missing, ref missing, ref missing,
   ref missing, ref missing, ref missing);

// Word beenden
object saveChanges = false;
word.Quit(ref saveChanges, ref missing, ref missing);
```

Listing 710: Ausdrucken und Speichern eines Dokuments und Beenden von Word

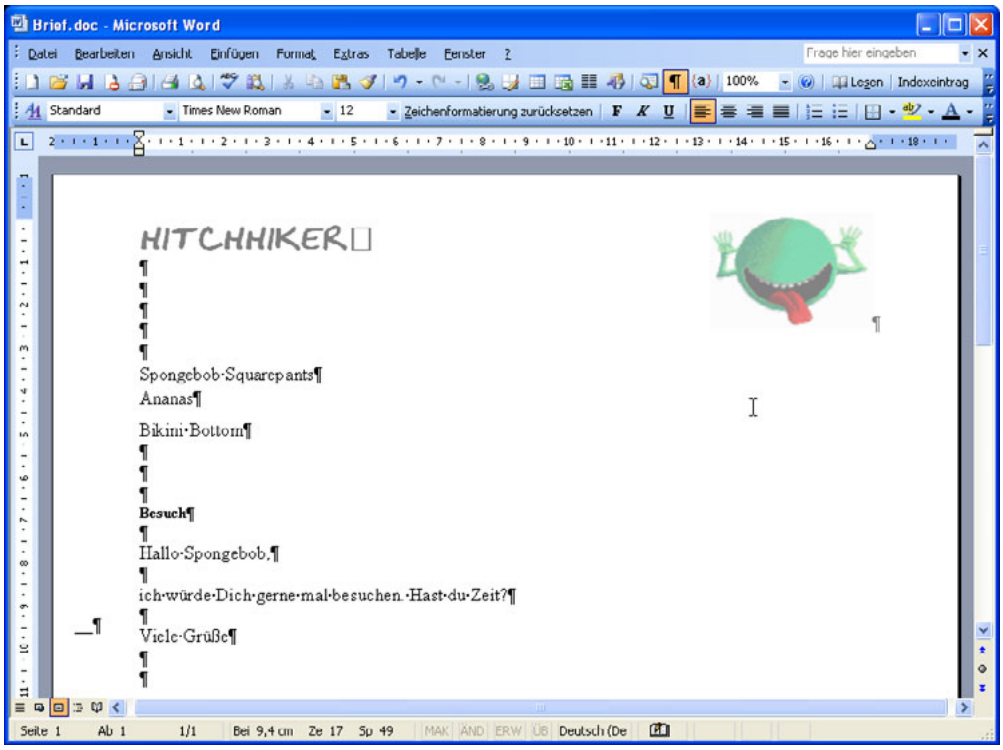


Abbildung 203: Das erzeugte Word-Dokument

353 Excel-Arbeitsmappen erzeugen

Excel kann ebenso einfach ferngesteuert werden wie Word (siehe Rezept 351). Zur Erzeugung von Excel-Arbeitsmappen benötigen Sie zunächst eine Instanz der Klasse `ApplicationClass` (die die Schnittstelle `Application` implementiert), die in der PIA-Assembly (siehe Rezept 350) *Microsoft.Office.Interop.Excel* verwaltet wird.

Hinweis

Das in diesem Rezept behandelte Fernsteuern von Excel ist nicht besonders schnell und außerdem problematisch, wenn das Ganze in einer Webanwendung ausgeführt wird. Die reduzierte Geschwindigkeit liegt daran, dass der Zugriff auf Excel über Prozessgrenzen hinweg über COM erfolgt. Dieses Problem können Sie über das Rezept 355 lösen, wenn Sie viele Daten lesen oder schreiben müssen. Sie können allerdings auch kommerzielle Komponenten verwenden, die Excel-Dokumente bearbeiten können. Eine (etwas eingeschränkt[1]) empfehlenswerte (kostenpflichtige) Komponente ist *Aspose.Cells* (*www.aspose.com/Products/Aspose.Cells*), die es erlaubt, Excel-Dokumente direkt (also ohne Excel) zu bearbeiten. Solche Komponenten lösen auch das Problem, das in Web

1. Aspose.Cells hat Probleme mit dem Drucken von Excel-Arbeitsmappen und dem Erzeugen von PDF-Dokumenten (über Aspose.Pdf). Das ausgedruckte bzw. erzeugte Dokument entspricht nicht 1:1 dem Originaldokument und erzeugt häufig zusätzliche Seiten, auf denen der nicht mehr passende Teil der vorherigen Seite zu sehen ist.

anwendungen entsteht, wenn Excel-Instanzen aufgrund von Fehlern in der Anwendung nicht geschlossen werden und folglich als Prozess im Speicher bleiben (und irgendwann den Webserver überlasten).

Um nun eine Excel-Arbeitsmappe zu erstellen, erzeugen Sie ein `Application`-Objekt oder referenzieren Sie eine vorhandene Excel-Instanz (siehe Rezept 350). Über die `Add`-Methode der `Workbooks`-Auflistung können Sie dann eine neue Arbeitsmappe erzeugen. Die `Workbooks`-Auflistung verwaltet alle in der Excel-Instanz geöffneten Arbeitsmappen.

Die `Add`-Methode erwartet ein (in COM optionales) `Object`-Argument, über das Sie die zu verwendende Vorlage spezifizieren können. Um eine normale Arbeitsmappe zu erzeugen, übergeben Sie an diesem Argument den Wert `System.Reflection.Missing.Value`. Anders als bei Word sind die Argumente der Excel-Methoden in der Regel nicht By Reference deklariert, weswegen Sie prinzipiell den nach `Object` konvertierten Wert `Missing.Value` direkt übergeben können. Um den Aufruf zu vereinfachen, verwende ich dafür aber eine Variable.

Um das folgende Beispiel kompilieren zu können müssen Sie die Excel-PIA-Assembly und (wegen der Verwendung der `MessageBox`-Klasse) die Assemblys *WindowsBase*, *PresentationCore* und *PresentationFramework* referenzieren und die Namensräume `System`, `System.IO`, `System.Reflection` und `System.Windows` importieren.

```
// Excel-Instanz erzeugen und sichtbar schalten
Microsoft.Office.Interop.Excel.Application excel =
   new Microsoft.Office.Interop.Excel.ApplicationClass();
excel.Visible = true;

// Arbeitsmappe ohne Vorlage erzeugen
object missing = Missing.Value;
excel.Workbooks.Add(missing);
```

Listing 711: Erzeugen einer Excel-Arbeitsmappe

Nun können Sie die aktive Arbeitsmappe referenzieren und über die `Add`-Methode der `Worksheets`-Auflistung ein Arbeitsblatt erzeugen. Da Excel so eingestellt werden kann, dass beim Start automatisch Arbeitsblätter erzeugt werden (per Voreinstellung sind das drei), können Sie zuvor abfragen, ob bereits Arbeitsblätter bestehen:

```
// Das aktive Workbook referenzieren
Microsoft.Office.Interop.Excel.Workbook workbook = excel.ActiveWorkbook;

// Arbeitsblatt erzeugen
if (workbook.Worksheets.Count == 0)
{
   workbook.Worksheets.Add(missing, missing, missing, missing);
}
```

Listing 712: Erzeugen eines Arbeitsblattes

Um mit dem ersten Arbeitsblatt zu arbeiten, sollten Sie dieses referenzieren. Dabei müssen Sie beachten, dass die `Worksheets`-Auflistung wie viele andere COM-Auflistungen auch 1-basiert ist. Über die `Name`-Eigenschaft können Sie das Arbeitsblatt gleich auch umbenennen:

```
// Das erste Arbeitsblatt referenzieren und umbenennen
Microsoft.Office.Interop.Excel.Worksheet worksheet =
   (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[1];
worksheet.Name = "Einkommensberechnung";
```

Listing 713: Referenzieren und Umbenennen des ersten Arbeitsblattes

Nun können Sie einzelne Zellen oder ganze Zellbereiche mit Text oder Formeln beschreiben und beliebig formatieren. Um herauszufinden, wie Sie einzelne Aktionen programmieren, können Sie in Excel einfach ein Makro aufzeichnen und den erzeugten Visual Basic-Programmcode nach C# konvertieren. Der Vorgang ist ähnlich dem, den ich im Rezept 351 bereits für Word beschrieben habe.

In Visual Basic würden Sie die `Range`-Eigenschaft des `Worksheet`-Objekts verwenden um direkt auf eine Zelle oder auf einen Zellbereich zuzugreifen. Diese spezielle Eigenschaft, der Sie als Index einen oder zwei Zellbezüge (in der in Excel üblichen Form *A1* oder *A1:C3*) als String übergeben können, kann in C# nicht direkt umgesetzt werden. Die Wrapper-Assembly enthält deswegen statt der Eigenschaft `Range` eine Methode `get_Range`, die ähnlich arbeitet. Dieser Methode übergeben Sie normalerweise am ersten Argument einen Zellbezug. Am zweiten Argument, das einen zweiten Zellbezug aufnimmt, können Sie `Missing.Value` übergeben. Sie erhalten dann ein `Range`-Objekt zurück, über das Sie die Zelle bzw. den Bereich mit einem Text beschreiben oder umformatieren können.

Zum Schreiben eines Textes können Sie in Visual Basic die `Value`-Eigenschaft verwenden, die aber (aus mir unbekannten Gründen) in der Wrapper-Assembly nicht direkt vorhanden ist und über die Methode `set_Value` repräsentiert wird. Alternativ können Sie aber auch die Eigenschaft `Value2` verwenden, die auch in der Wrapper-Assembly implementiert ist.

Das folgende Beispiel schreibt zunächst einen Text in die Zelle A1 und formatiert dann deren Schriftart um. Dann werden noch die Zellen A2 bis A5 mit Texten beschrieben:

```
// Die Zelle A1 referenzieren, beschreiben und die Schriftart definieren
Microsoft.Office.Interop.Excel.Range range =
   worksheet.get_Range("A1", missing);
range.set_Value(missing, "Einkommens-Berechnung");
// Alternative range.Value2 = "Einkommens-Berechnung";
range.Font.Size = 16;
range.Font.Bold = true;

// A2 bis A5 beschreiben
worksheet.get_Range("A2", missing).Value2 = "2004";
worksheet.get_Range("A3", missing).Value2 = "2005";
```

Listing 714: Beschreiben und Formatieren von Zellen

```
worksheet.get_Range("A4", missing).Value2 = "2006";
worksheet.get_Range("A5", missing).Value2 = "2007";
```

Listing 714: Beschreiben und Formatieren von Zellen (Forts.)

Über die `Cells`-Eigenschaft können Sie über Integer-Indizes auf Zellen zugreifen, z. B. um diese in einer Schleife zu beschreiben oder auszulesen. Am ersten Argument übergeben Sie den (1-basierten) Index der Zeile, am zweiten den Index der Spalte. `Cells` gibt wie `get_Range` ein Range-Objekt zurück, über das Sie den Inhalt und die Formatierung der Zelle erreichen. Das folgende Beispiel schreibt den Inhalt eines Arrays in die Zellen B2 bis B5:

```
double[] sales = {120000, 118000, 130000, 150000};
for (int i = 0; i < 4; i++)
{
   int row = i + 2;
   int col = 2;
   worksheet.Cells[row, col] = sales[i];
}
```

Listing 715: Zugriff auf Zellen über Integer-Indizes

Formeln können Sie über die `Formula`- oder `FormularLocal`-Eigenschaft eines `Range`-Objekts erzeugen. Dazu verwenden Sie die Syntax, die auch in Excel direkt eingesetzt wird.

Hinweis

Laut der Dokumentation verwendet die `Formula`-Eigenschaft die englische Formelsyntax, für die lokale Variante müssten Sie die `FormulaLocal`-Eigenschaft verwenden. Bei der Fernsteuerung von Excel über ein Visual-Basic-6-Programm funktioniert das auch. Eigenartigerweise konnte ich in meinen Versuchen auch mit der `Formula`-Eigenschaft nur die lokale (deutsche) Formelvariante verwenden.

```
// Eine Formel in die Zelle B6 schreiben und die Zelle fett formatieren
worksheet.get_Range("B6", missing).FormulaLocal = "=SUMME(B2:B5)";
worksheet.get_Range("B6", missing).Font.Bold = true;
```

Listing 716: Erzeugen einer Formel

Statt direkte Bezüge für die Formel zu verwenden, könnten Sie auch über die `FormulaR1C1`-Eigenschaft Formeln mit Bezügen im so genannten Z1S1-Bezugssystem (in Englisch: R1C1-System) übergeben. Dabei geben Sie die Zeile und die Spalte jeweils relativ zur aktuellen Zelle über einen Index in runden (bzw. in der englischen Syntax in eckigen) Klammern an. *Z(-4)S:Z(-1)S* bedeutet z. B.: »Von der Zelle in der aktuellen Zeile -4 und der aktuellen Spalte bis zur Zelle in der aktuellen Zeile -1 und der aktuellen Spalte«. Damit könnten Sie Formeln auch für dynamisch erzeugte Zeilen relativ einfach erzeugen:

```
worksheet.get_Range("B6", missing).FormulaR1C1Local = "=SUMME(Z(-4)S:Z(-1)S)";
```

Hinweis

Wie schon bei der `Formula`-Eigenschaft sollte auch `FormulaR1C1` die Formel eigentlich in der englischen Syntax verwalten. Bei meinen Versuchen war dies aber unter Excel XP und 2003 nicht möglich. Ich musste auch der `FormulaR1C1`-Eigenschaft die deutsche Syntax übergeben.

Hinweis

Wenn die Formel einen Fehler enthält (bzw. in der falschen Sprachversion übergeben wurde), resultiert dies normalerweise in einer wenig aussagekräftigen Ausnahme vom Typ `COMException` mit der Meldung »Ausnahme von HRESULT: 0x800A03EC«.

Nun können Sie das erzeugte Arbeitsblatt noch ausdrucken, wozu Sie die `PrintOut`-Methode des `Worksheet`-Objekts verwenden. Diese Methode arbeitet mit einigen optionalen Argumenten, über die Sie u. a. den Seitenbereich, die Anzahl der Kopien und den Drucker spezifizieren können. Da ich davon ausgehe, dass normalerweise das ganze Dokument auf dem Standarddrucker gedruckt wird, definiere ich nur die Anzahl der Kopien und übergebe an den anderen Argumenten den Wert `Missing.Value`.

```
object copies = 1;
worksheet.PrintOut(missing, missing, copies, missing, missing, missing,
   missing, missing);
```

Listing 717: Ausdrucken eines Arbeitsblatts

Zum Speichern einer Arbeitsmappe können Sie deren `SaveAs`-Methode aufrufen. Diese Methode besitzt eine Vielzahl an optionalen `Object`- und ein eigentlich ebenfalls optionales Argument mit dem definierten Typ `Excel.XlSaveAsAccessMode`. Am ersten Argument übergeben Sie den Dateinamen. Am zweiten Argument können Sie das Dateiformat in der Form eines Wertes der Aufzählung `Excel.XlFileFormat` bestimmen. Dieses Argument ist interessant, wenn Sie die Datei z. B. in einem älteren Excel-Format speichern wollen, damit ältere Excel-Versionen darauf zugreifen können. Das Beispiel speichert die Datei im Excel 95/97-Format. Die weiteren Argumente, über die Sie u. a. ein Passwort vergeben können, sind relativ uninteressant und können mit `Missing.Value` belegt werden. Am siebten Argument erwartet `SaveAs` dummerweise keinen `Object`-Typ, sondern einen Wert der Aufzählung `Excel.XlSaveAsAccessMode`. Dieses Argument bestimmt, ob mehrere Benutzer gleichzeitig auf die Excel-Arbeitsmappe zugreifen können oder nicht. Übergeben Sie hier idealerweise `Excel.XlSaveAsAccessMode.xlNoChange` um die in der Arbeitsmappe bereits vorgenommene Einstellung nicht zu ändern.

`SaveAs` besitzt ein kleines Manko: Falls die Datei bereits existiert, haben Sie keine Möglichkeit, festzulegen, dass diese überschrieben wird. Excel fragt dann immer automatisch nach, ob die Datei überschrieben werden soll, und erzeugt eine Ausnahme vom Typ `COMException`, wenn der Anwender nicht den Ja-Schalter betätigt. Um diesen automatischen Dialog zu vermeiden, sollten Sie selbst abfragen, ob die Datei bereits existiert, und diese gegebenenfalls zuvor löschen, wie ich es in Listing 718 implementiert habe.

```
// Dateiname definieren
string appPath = System.IO.Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string fileName = Path.Combine(appPath, "Einkommensberechnung.xls");

// Abfragen, ob die Datei bereits existiert
bool writeFile = true;
if (File.Exists(fileName))
{
   writeFile = false;
   if (MessageBox.Show("Die Datei '" + fileName + "' existiert " +
      "bereits.\r\n\r\nWollen Sie diese Datei überschreiben?",
      "Excel-Arbeitsmappen erzeugen", MessageBoxButton.YesNo,
      MessageBoxImage.Question) == MessageBoxResult.Yes)
   {
      // Datei löschen
      File.Delete(fileName);
      writeFile = true;
   }
   else
   {
      // Die Arbeitsmappe schließen ohne diese zu speichern
      workbook.Close(false, missing, missing);
   }
}

if (writeFile)
{
   // Die Arbeitsmappe im Excel-95/97-Format speichern
   object fileFormat =
      Microsoft.Office.Interop.Excel.XlFileFormat.xlExcel9795;
   workbook.SaveAs(fileName, fileFormat, missing, missing, missing, missing,
      Microsoft.Office.Interop.Excel.XlSaveAsAccessMode.xlNoChange,
      missing, missing, missing, missing, missing);
}

// Excel beenden
excel.Quit();
```

Listing 718: Speichern einer Excel-Arbeitsmappe und Beenden von Excel



Abbildung 204: Die erzeugte Excel-Arbeitsmappe

354 Daten in Excel-Arbeitsmappen erweitern

Excel-Arbeitsmappen werden häufig verwendet, um Daten tabellarisch zu verwalten und über diese Daten Berechnungen auszuführen. Wenn Sie diese Daten programmgesteuert erweitern müssen, können Sie, nachdem Sie die Arbeitsmappe geöffnet und das Arbeitsblatt referenziert haben, einfach die letzte Zeile suchen, die noch Daten speichert, und hinter diese Zeile neue Zeilen für die neuen Daten einfügen.

Die folgenden Listings zeigen, wie das programmiert wird. Als Basis verwende ich die Excel-Datei, die in Rezept 353 erstellt wurde. In diesem Rezept behandle ich auch die Grundlagen der Fernsteuerung von Excel. Beachten Sie auch die Hinweise zur Performance und zur Ausführung in Webanwendungen in Rezept 353.

Zunächst müssen Sie die Arbeitsmappe öffnen und das Arbeitsblatt referenzieren. Dazu erzeugen Sie eine Excel-Instanz und rufen die `Open`-Methode der `Workbooks`-Auflistung auf. `Open` erwartet am ersten Argument den Dateinamen (als String). Die weiteren Argumente, die alle optional sind, bestimmen u. a., ob Links in der Datei automatisch aktualisiert werden sollen, ob die Datei schreibgeschützt geöffnet werden soll, und geben ein eventuelles Passwort an. Da diese Argumente in der Praxis meist unerheblich sind, übergebe ich einfach `Missing.Value`. Die `Open`-Methode gibt eine Referenz auf die geöffnete Arbeitsmappe zurück, die Sie idealerweise einer Variablen vom Typ `Workbook` zuweisen.

Dann referenzieren Sie das anzusprechende Arbeitsblatt über die `Worksheets`-Auflistung des `Workbook`-Objekts. Dieser Auflistung können Sie den Index oder den Namen des Arbeitsblattes übergeben. Listing 719 übergibt den Namen.

Zum Kompilieren dieses Programms müssen Sie die Excel-PIA-Assembly referenzieren (siehe Rezept 350 für die Word-PIA-Assembly) und die Namensräume `System`, `System.IO` und `System.Reflection` einbinden.

```
// Excel-Instanz erzeugen und sichtbar schalten
Microsoft.Office.Interop.Excel.Application excel =
   new Microsoft.Office.Interop.Excel.ApplicationClass();
excel.Visible = true;

// Arbeitsmappe öffnen
object missing = Missing.Value;
string fileName = Path.Combine(appPath, "Einkommensberechnung.xls");
Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(
   fileName, missing, missing, missing, missing, missing, missing, missing,
   missing, missing, missing, missing, missing, missing, missing);

// Arbeitsblatt 'Einkommensberechnung' referenzieren
Microsoft.Office.Interop.Excel.Worksheet worksheet =
   (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[
   "Einkommensberechnung"];
```

Listing 719: Öffnen einer Excel-Arbeitsmappe und Referenzieren eines Arbeitsblatts

Zum Suchen der ersten Zeile, die keine Daten mehr enthält, können Sie die Zeilen in einer Schleife durchgehen und den Wert der Zellen in der Zeile über die `Cells`-Eigenschaft abfragen, der Sie am ersten Argument den Zeilen- und am zweiten Argument den Spaltenindex übergeben. Gibt die `Value2`-Eigenschaft einer Zelle `null` zurück, ist diese Zelle leer. Listing 720 geht die Zeilen ab der ersten Zeile durch und fragt den Wert der Zelle in der ersten Spalte ab. Zur Vereinfachung referenziert das Programm die einzelnen Zellen in der Variable *range*. Da `Cells` eine `object`-Referenz zurückgibt, muss diese in den Datentyp `Excel.Range` konvertiert werden. Enthält eine Zelle den Wert `null`, bricht das Programm die Schleife ab. Da nun die erste Zelle in der Zeile referenziert ist, die der letzten Datenzeile folgt, referenziert Listing 720 dann über die `get_Offset`-Methode die Zelle oberhalb der gefundenen. `get_Offset` erwartet einen Offset-Index für die Zeile und die Spalte und gibt eine Referenz auf das `Range`-Objekt zurück, das diesem Offset entspricht.

```
int row = 0;
Microsoft.Office.Interop.Excel.Range range = null;
do
{
   row++;
   range = (Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row, 1];
   if (range.Value2 == null)
   {
      break;
   }
```

Listing 720: Suchen der ersten Zeile in einem Excel-Arbeitsblatt, die keine Daten mehr enthält

```
} while (true);

// Die erste Zelle der letzten Datenzeile referenzieren
range = (Microsoft.Office.Interop.Excel.Range)worksheet.Cells[row - 1, 1];
```

Listing 720: Suchen der ersten Zeile in einem Excel-Arbeitsblatt, die keine Daten mehr enthält (Forts.)

Diese Methode, die letzte Datenzeile zu finden, ist sehr flexibel, da Sie auch mehrere Zellen daraufhin überprüfen können, ob diese noch Daten enthalten. Wenn Sie sich darauf verlassen können, dass eine Spalte immer Daten enthält und ein Leerwert in dieser Spalte das Ende der Daten kennzeichnet, können Sie auch die folgende, etwas effizientere Lösung einsetzen:

```
Microsoft.Office.Interop.Excel.Range range = worksheet.get_Range("A1",
   missing).get_End(Microsoft.Office.Interop.Excel.XlDirection.xlDown);
```

Listing 721: Effizienteres Ermitteln der letzten Datenzeile

Diese Lösung ruft, ausgehend von der Zelle A1, die `get_End`-Methode unter der Übergabe von `Excel.XlDirection.xlDown` auf. `get_End` gibt dann eine Referenz auf die letzte Zelle zurück, die noch Daten verwaltet. Damit erreichen Sie programmgesteuert das, was Sie in Excel auch durch die Betätigung von Strg + Pfeil u erreichen. Den entsprechenden Programmcode habe ich auch einfach über die Aufzeichnung eines Makros in Excel ermittelt ☺.

Nun, da die Zelle in der ersten Spalte der letzten Datenzeile referenziert ist, könnten Sie die Zelle unterhalb der gefundenen referenzieren und dort eine neue Zeile einfügen (Excel fügt neue Zeilen standardmäßig oberhalb ein). Leider führt das aber zu einem Problem: Auf diese Weise werden die Zellbezüge von Formeln, die sich auf den Datenbereich beziehen (im Beispiel ist das die Formel `SUMME(B2:B5)`) nicht automatisch erweitert. Sie müssten also alle Formeln überarbeiten, was natürlich nicht sinnvoll ist.

Die Lösung dieses Problems ist, dass Sie einfach eine Zeile oberhalb der letzten Datenzeile einfügen und die Daten der letzten Zeile in diese neue Zeile umkopieren. In diesem Fall passt Excel alle Zellbezüge automatisch an. Listing 722 setzt dies für zwei neue Zeilen um. Dabei wird zunächst über die `Insert`-Methode des `Range`-Objekts, das die gesamte aktuelle Zeile darstellt, eine neue Zeile eingefügt. Die gesamte Zeile erhalten Sie über die `EntireRow`-Eigenschaft eines `Range`-Objekts. Normalerweise müsste dann immer noch die erste Zelle der aktuellen Zeile referenziert sein. Da dies aber beim zweiten Einfügen einer Zeile nicht mehr der Fall war, referenziert Listing 722 zur Sicherheit die erste Zelle explizit (über die `Cells`-Eigenschaft der gesamten Zeile).

Dann kopiert das Programm die Daten der zuvor letzten Datenzeile nach oben in die neue Zeile und fügt die neuen Daten in die aktuelle Zeile ein. Das Ganze wird schließlich noch für eine weitere neue Zeile wiederholt.

```
// Eine Zeile einfügen
range.EntireRow.Insert(missing, missing);

// Die erste Zelle in der aktuellen Zeile referenzieren (zur Sicherheit)
range = (Microsoft.Office.Interop.Excel.Range)range.EntireRow.Cells[1, 1];

// Die Daten der aktuellen Zeile in die neue oben kopieren
range.get_Offset(-1, 0).Value2 = range.Value2;
range.get_Offset(-1, 1).Value2 = range.get_Offset(0, 1).Value2;

// Daten in die aktuelle Zeile einfügen
range.Value2 = 2008;
range = range.get_Offset(0, 1);
range.Value2 = 185000;

// Eine weitere Zeile einfügen, Daten kopieren und neue Daten einfügen
range.EntireRow.Insert(missing, missing);
range = (Microsoft.Office.Interop.Excel.Range)range.EntireRow.Cells[1, 1];
range.get_Offset(-1, 0).Value2 = range.Value2;
range.get_Offset(-1, 1).Value2 = range.get_Offset(0, 1).Value2;
range.Value2 = 2009;
range = range.get_Offset(0, 1);
range.Value2 = 221000;
```

Listing 722: Einfügen von zwei neuen Zeilen in das Arbeitsblatt

Nun können Sie noch ausdrucken und die Arbeitsmappe speichern. Grundlagen zum Drucken finden Sie im Rezept 353. Speichern können Sie einfach über die `Save`-Methode des `Workbook`-Objekts:

```
// Arbeitsmappe speichern
workbook.Save();

// Excel beenden
excel.Quit();
```

Listing 723: Speichern der Arbeitsmappe und Beenden von Excel

Abbildung 205 zeigt die erweiterte Arbeitsmappe in Excel.

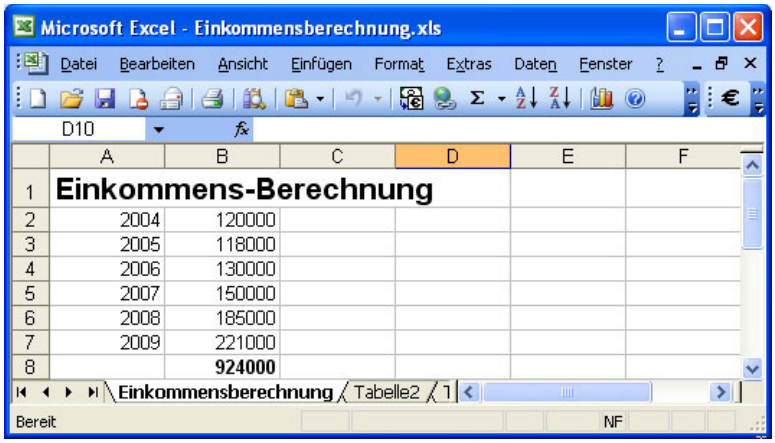


Abbildung 205: Die erweiterte Excel-Arbeitsmappe

355 Performantes Lesen und Schreiben in Excel-Arbeitsmappen

Wenn Sie, wie ich es in den Rezepten 353 und 354 erläutert habe, auf die Zellen eines Excel-Arbeitsplatzes einzeln zugreifen, ist das Lesen und Schreiben nicht besonders performant. Der Grund dafür ist, dass jeder Zugriff über das COM-Modell und über Prozessgrenzen hinweg erfolgt. Wenn Sie viele Daten lesen und/oder schreiben, kann es vorkommen, dass der Zugriff auf die Daten mehrere Minuten in Anspruch nimmt.

Ein Leser der ersten Auflage des C# Codebook, Konstantin Norin, hat mich darauf aufmerksam gemacht, dass das Lesen und Schreiben auch wesentlich performanter möglich ist. Sie können nämlich einen Bereich in einem Rutsch in ein zweidimensionales Array lesen oder mit den Werten eines solchen Arrays beschreiben.

Um einen Bereich komplett einzulesen, casten Sie die Rückgabe der `Values2`-Eigenschaft eines `Range`-Objekts in ein zweidimensionales `object`-Array:

```
object[,] values =
   (object[,])worksheet.get_Range("A1", "A1000").Value2;
```

Die Dimensionen dieses Arrays beginnen, wie im COM-Modell üblich, bei 1. Beim Durchgehen sollten Sie sicherheitshalber über die Methoden `GetLowerBound` und `GetUpperBound` die Grenzen der jeweiligen Dimension ermitteln.

Das folgende Beispiel liest die Excel-Datei *Demo.xls* aus dem Anwendungsordner ein, liest den Inhalt des Bereichs A1 bis A1000 in ein Array und verdoppelt schließlich den (Integer-)Wert der Zellen im Array. Es benötigt die Referenzierung der Excel-PIA-Assembly (siehe Rezept 350) und den Import der Namensräume `System`, `System.IO` und `System.Reflection`.

```
// Excel-Instanz erzeugen und sichtbar schalten
Microsoft.Office.Interop.Excel.Application excel =
   new Microsoft.Office.Interop.Excel.ApplicationClass();
excel.Visible = true;

// Arbeitsmappe einlesen
string appPath = System.IO.Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string filename = Path.Combine(appPath, "Demo.xls");
Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(
   filename, Missing.Value, Missing.Value, Missing.Value, Missing.Value,
   Missing.Value, Missing.Value, Missing.Value, Missing.Value,
   Missing.Value, Missing.Value, Missing.Value, Missing.Value,
   Missing.Value, Missing.Value);

// Das erste Arbeitsblatt referenzieren
Microsoft.Office.Interop.Excel.Worksheet worksheet =
   (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[1];

// Den Wert der Zellen A1 bis A1000 in ein zweidimensionales Array lesen
//Array values = (Array)worksheet.get_Range("A1", "A1000").Value2;
object[,] values = (object[,])worksheet.get_Range("A1", "A1000").Value2;

// Den Wert jeder Zelle zu Demozwecken verdoppeln
for (int i = values.GetLowerBound(0); i <= values.GetUpperBound(0); i++)
{
   int newValue = Convert.ToInt32(values[i, 1]) * 2;
   values[i, 1] = newValue;
}
```

Listing 724: Performantes Lesen eines Excel-Bereichs

Beim Durchgehen des Arrays müssen Sie natürlich darauf achten, dass die einzelnen »Zellen« beliebige Werte speichern können. Im Beispiel habe ich allerdings darauf verzichtet.

Das Schreiben funktioniert dann lediglich umgekehrt. Das Beispiel schreibt die geänderten Werte in die Zellen B1 bis B1000:

```
// Das Array in die Zellen B1 bis B1200 zurückschreiben
worksheet.get_Range("B1", "B1200").Value2 = values;
```

Listing 725: Performantes Schreiben eines Excel-Bereichs

Beim Schreiben sollte das Array natürlich zu dem zu beschreibenden Bereich passen. Ist dies nicht der Fall, reagiert Excel fehlertolerant: Der Bereich wird immer ausgehend von der Start-Zelle bis maximal zur End-Zelle beschrieben. Sind mehr Daten im Array, als der Bereich zulässt, werden die über den Bereich hinausgehenden Zellen nicht überschrieben. Sind weniger Daten im Array als im Bereich, schreibt Excel *#NV* in die Zellen.

356 Kontakte aus Outlook auslesen

Outlook wird von vielen Anwendern zur Verwaltung ihrer Kontakte und als E-Mail-Client eingesetzt. Die Kontakte können Sie natürlich über die Fernsteuerung von Outlook auslesen, z. B. um dem Anwender in Ihrem Programm eine Auswahl der Kontakte anzubieten und damit einen Serienbrief zu generieren.

Dazu referenzieren Sie zunächst die PIA-Assembly *Microsoft.Office.Interop.Outlook* (siehe Rezept 350). Dann erzeugen Sie wie bei allen anderen Office-Anwendungen eine Instanz der Klasse `Outlook.ApplicationClass`. Alternativ können Sie auch eine vorhandene Outlook-Instanz referenzieren, wie ich es prinzipiell im Rezept 350 beschreibe. Gerade bei Outlook ist diese Technik aufgrund des relativ langsamen Starts sehr interessant. Ein Sichtbar-Schalten ist allerdings bei Outlook nicht möglich.

Zum Zugriff auf die Outlook-Ordner benötigen Sie eine Referenz auf die aktuelle MAPI-Sitzung (Outlook verwendet das Messaging API zur Verwaltung der Ordner und für E-Mail-Funktionalitäten), die Sie über die Eigenschaft `Session` erreichen. Diese Eigenschaft besitzt den Typ `Outlook.NameSpace`. Sofern der Zugang zu Outlook geschützt ist, können Sie das Programm über die `Logon`-Methode des `NameSpace`-Objekts authentifizieren:

```
Microsoft.Office.Interop.Outlook.Application outlook =
   new Microsoft.Office.Interop.Outlook.Application();
Microsoft.Office.Interop.Outlook.NameSpace session = outlook.Session;
string profileName = "Outlook";
string password = null;
bool showDialog = true;
bool newSession = false;
session.Logon(profileName, password, showDialog, newSession);
```

Listing 726: Erzeugen einer Outlook-Instanz, Referenzieren der aktuellen Sitzung und Logon in Outlook

Den Default-Kontakte-Ordner können Sie sehr einfach über die `GetDefaultFolder`-Methode des `NameSpace`-Objekts ermitteln, indem Sie als Argument den Wert `Outlook.OlDefaultFolders.olFolderContacts` übergeben. Sie erhalten eine Referenz auf ein `MAPIFolder`-Objekt zurück. Ein solches Objekt repräsentiert einen beliebigen Ordner (auch z. B. E-Mail- oder Kalender-Ordner). Über die Eigenschaft `Items`, die eine normale COM-Auflistung (mit dem Startindex 1) ist, erreichen Sie die in einem Ordner gespeicherten Elemente. Dabei müssen Sie ein wenig aufpassen, denn ein Ordner kann beliebige Elemente speichern. Sie können nicht davon ausgehen, dass ein Kontakte-Ordner nur Elemente des Typs `Outlook.ContactItem` verwaltet. Deswegen sollten Sie das Lesen der Elemente wie in Listing 727 gezeigt vornehmen:

```
Microsoft.Office.Interop.Outlook.MAPIFolder folder =
   session.GetDefaultFolder(Microsoft.Office.Interop.Outlook.
   OlDefaultFolders.olFolderContacts);
```

Listing 727: Auslesen der Kontakte im Outlook-Default-Kontakte-Ordner

```
Console.WriteLine("{0} Kontakte im Ordner {1}",
   folder.Items.Count, folder.Name);
for (int i = 1; i <= folder.Items.Count; i++)
{
   Microsoft.Office.Interop.Outlook.ContactItem contact =
      folder.Items[i] as Microsoft.Office.Interop.Outlook.ContactItem;
   if (contact != null)
   {
      Console.WriteLine("{0} {1} {2} {3}, {4} {5}", contact.CompanyName,
         contact.FirstName, contact.MiddleName, contact.LastName,
         contact.HomeAddressPostalCode, contact.HomeAddressCity);
   }
}
```

Listing 727: Auslesen der Kontakte im Outlook-Default-Kontakte-Ordner (Forts.)

Ein Outlook-Kontakt-Element besitzt eine Vielzahl an Eigenschaften, die Sie ja wahrscheinlich prinzipiell von Outlook her bereits kennen. Ich verzichte hier auf eine Beschreibung, da die Namen der Eigenschaften größtenteils selbsterklärend sind. Eine recht mager ausgefallene Referenz der Outlook-2003-Objekte finden Sie, wenn Sie an der Adresse *msdn.microsoft.com/library* nach »Outlook Object Model« suchen (*msdn.microsoft.com/en-us/library/aa221870.aspx).* Das Outlook-2000-Objektmodell, das sich nicht allzu sehr von dem neuen Modell unterscheidet, finden Sie übrigens an der Adresse *msdn.microsoft.com/library/en-us/modcore/html/deovrMicrosoftOutlook2000.asp.*

Wenn Sie auf spezielle Ordner zugreifen wollen, müssen Sie diese einzeln ausgehend vom Stammordner referenzieren. Handelt es sich um den Unterordner eines anderen Ordners, erreichen Sie diesen über die Eigenschaft `Folders` eines `MAPIFolder`-Objekts. Das folgende Beispiel zeigt, wie Sie den Ordner *Persönliche Ordner/Kontakte/Handy* referenzieren.

```
string folderName1 = "Persönliche Ordner";
string folderName2 = "Kontakte";
string folderName3 = "Handy";
try
{
   folder = (Microsoft.Office.Interop.Outlook.MAPIFolder)
      session.Folders[folderName1];
   folder = folder.Folders[folderName2];
   folder = folder.Folders[folderName3];

   Console.WriteLine();
   Console.WriteLine("{0} Kontakte im Ordner {1}", folder.Items.Count,
      folder.Name);
   for (int i = 1; i <= folder.Items.Count; i++)
   {
```

Listing 728: Auslesen eines speziellen Kontakte-Ordners

```
      Microsoft.Office.Interop.Outlook.ContactItem contact =
         folder.Items[i] as Microsoft.Office.Interop.Outlook.ContactItem;
      if (contact != null)
      {
         Console.WriteLine("{0}", contact.CompanyAndFullName);
      }
   }
}
catch (Exception ex)
{
   Console.WriteLine("Fehler: " + ex.Message);
   outlook.Quit();
   return;
}
```

Listing 728: Auslesen eines speziellen Kontakte-Ordners (Forts.)

Das Beispiel nutzt übrigens die Eigenschaft `CompanyAndFullName`, die den Namen der Firma und den vollen Namen des Kontakts in einer gut formatierten Form zurückgibt.

Wie in Listing 728 sollten Sie für den Fall, dass der Ordner nicht existiert, eine Ausnahmebehandlung vorsehen. Die Fehlermeldung im Fall eines nicht existenten Ordners (»Der Vorgang konnte nicht ausgeführt werden. Ein Objekt wurde nicht gefunden«) ist so einigermaßen aussagekräftig.

Schließlich zeige ich noch, wie Sie die einzelnen Kontakte-Ordner einer Outlook-Instanz durchgehen können, um diese mit den enthaltenen Kontakten z. B. in einem `TreeView`-Steuerelement darzustellen. Dazu benötigen Sie eine Methode zum rekursiven Durchgehen der Kontakte-Ordner:

```
public static void ReadFolderItems(Microsoft.Office.Interop.Outlook.
   MAPIFolder folder)
{
   // Zunächst alle Elemente auslesen
   Console.WriteLine();
   Console.WriteLine(folder.FullFolderPath);
   for (int i = 1; i <= folder.Items.Count; i++)
   {
      Microsoft.Office.Interop.Outlook.ContactItem contact =
         folder.Items[i] as Microsoft.Office.Interop.Outlook.ContactItem;
      if (contact != null)
      {
         Console.WriteLine(contact.CompanyAndFullName);
      }
   }
```

Listing 729: Methode zum rekursiven Durchgehen der Outlook-Kontakte-Ordner

```
   // Dann alle Unterordner durchgehen, die selbst wieder Kontakte-Ordner sind
   for (int i = 1; i <= folder.Folders.Count; i++)
   {
      Microsoft.Office.Interop.Outlook.MAPIFolder
         subFolder = folder.Folders[i];
      if (subFolder.DefaultItemType == Microsoft.Office.Interop.Outlook.
         OlItemType.olContactItem)
      {
         // Methode rekursiv aufrufen
         ReadFolderItems(subFolder);
      }
   }
}
```

Listing 729: Methode zum rekursiven Durchgehen der Outlook-Kontakte-Ordner (Forts.)

Die Methode, der eine Referenz auf ein `MAPIFolder`-Objekt übergeben wird, liest zunächst wie bereits bekannt alle Kontakte aus diesem Ordner aus. Dann geht sie die Unterordner des Ordners durch, die über die Eigenschaft `Folders` erreichbar sind. Um zu überprüfen, ob es sich um einen Kontakte-Ordner handelt, vergleicht `ReadFolderItems` den Default-Elementtyp der einzelnen Ordner mit der Konstante `Outlook.OlItemType.olContactItem`. Dabei gehe ich davon aus, dass Ordner, deren Default-Elementtyp Kontakte sind, auch Kontakte-Ordner sind ☺. Wird ein untergeordneter Kontakte-Ordner gefunden, ruft die Methode sich mit diesem Ordner wieder auf. Im Programm müssen Sie `ReadFolderItems` lediglich mit einem Wurzel-Ordner aufrufen:

```
string rootFolderName = "Persönliche Ordner";
ReadFolderItems((Outlook.MAPIFolder)session.Folders[rootFolderName]);
```

Am Ende Ihrer Arbeit sollten Sie nicht vergessen, die Outlook-Instanz freizugeben, da der Prozess ansonsten bestehen bleibt und den Arbeitsspeicher belastet:

```
outlook.Quit();
```

Reflection und Serialisierung

357 Informationen über den Aufrufer in einer Methode oder einem Konstruktor ermitteln

Tritt in einer komplexen Anwendung innerhalb einer Methode ein Fehler auf, ist es häufig sinnvoll zu wissen, welche andere Methode diese Methode aufgerufen hat. Wenn der Fehler auf Ihrem Entwicklungssystem auftritt, können Sie beim Debuggen die Aufrufliste zu Rate ziehen. Tritt der Fehler aber nur auf einem Anwender-Rechner auf, wird das Debugging schon schwieriger. In diesem Fall ist es sinnvoll, Informationen über den Aufrufer der Methode auszulesen und in der Fehlermeldung auszugeben.

Dazu können Sie einfach die `StackTrace`-Eigenschaft des `Exception`-Objekts auslesen, das Sie im `catch`-Block einer Ausnahmebehandlung erhalten. Sie können aber auch direkt den Namen der aufrufenden Methode auslesen. Sinnvoll ist das ebenfalls, wenn Sie in einer Logdatei protokollieren wollen, wie oft und von welcher anderen Methode eine Methode aufgerufen wird.

Um den Namen der aufrufenden Methode herauszufinden, können Sie einfach ein neues `StackTrace`-Objekt erzeugen (aus dem Namensraum `System.Diagnostics`). Ein solches Objekt verwaltet `StackFrame`-Instanzen, die Informationen zu den einzelnen Aufrufen in der Aufrufliste enthalten. Über die `GetFrame`-Methode, der Sie einen Index übergeben, können Sie die einzelnen Stack-Frames auslesen. Über deren `GetMethod`-Methode erhalten Sie ein `MethodBase`-Objekt mit Informationen zur Methode. Über deren Eigenschaft `Name` erhalten Sie den Namen, die Eigenschaft `ReflectedType` gibt Informationen zur Klasse oder Struktur, in der die Methode gespeichert ist.

Listing 730 zeigt, wie Sie mit dieser Technik den Namen und die Klasse der aufrufenden Methode herausfinden. Das Beispiel liest den Stack-Frame mit dem Index 1 aus (der Stack-Frame mit dem Index 0 ist der der Methode `Test` selbst).

Das Beispiel erfordert den Import des Namensraums `System.Diagnostics`.

```
public static void Test()
{
   // Den Namen und die Klasse der aufrufenden Methode ermitteln
   StackTrace stackTrace = new StackTrace();
   string callerMethodName = stackTrace.GetFrame(1).GetMethod().Name;
   string callerClassName =
      stackTrace.GetFrame(1).GetMethod().ReflectedType.Name;

   Console.WriteLine("Aufrufende Methode: {0}", callerMethodName);
   Console.WriteLine("Aufrufende Klasse: {0}", callerClassName);
}
```

Listing 730: Ermittlung des Namens und des Klassennamens der aufrufenden Methode



Abbildung 206: Das Beispielprogramm in Aktion

358 Programmcode dynamisch ausführen

Wenn Sie einmal Programmcode dynamisch ausführen müssen (z. B. zur Programmierung eines einfachen Rechners, in dem der Anwender eine Rechenoperation als String eingeben kann), können Sie einen komplizierten und einen einfachen Weg gehen. Der komplizierte ist die Verwendung der Klassen im Namensraum `System.Reflection.Emit`. Diese Klassen ermöglichen die dynamische Erzeugung von Assemblys. Leider müssen Sie hier aber CIL-Code angeben, was für die Praxis also kaum zu gebrauchen ist. Besser wäre es, den C#-Quellcode für eine Assembly in einen String speichern und diesen in eine Assembly kompilieren zu können, die dann schließlich im Programm per Reflection ausgeführt wird. Und das ist kein Problem. Sie müssen lediglich wissen, wie es geht. Und das zeige ich hier ☺.

Über eine Instanz der `CodeDomProvider`-Klasse aus dem Namensraum `System.CodeDom.Compiler` können Sie beliebigen Quellcode in einer von .NET unterstützten Sprache in eine Assembly kompilieren und die erzeugten Methoden dynamisch ausführen. Die Assembly muss dazu nicht als Datei erzeugt werden, sondern kann lediglich im Speicher existieren. Der Quellcode muss natürlich den Regeln von .NET entsprechen, sollte also einen Namensraum und in diesem zumindest eine Klasse mit einer Methode beinhalten. Bei der dynamischen Ausführung der Methode können Sie (natürlich) Argumente übergeben und einen eventuellen Rückgabewert auswerten.

Wie Sie dies programmieren, zeige ich an einem einfachen Beispiel, das den Import der Namensräume `System`, `System.CodeDom.Compiler` und `System.Reflection` erfordert.

Zunächst müssen Sie natürlich den Quellcode zusammenstellen. Um dieses Rezept so einfach wie möglich zu halten, verwende ich eine Methode, die lediglich zwei Zahlen addiert und das Ergebnis zurückgibt:

```
string assemblyNamespace = "DynamicCode";
string className = "Demo";
string methodName = "Add";
string source = "namespace " + assemblyNamespace +
   "{" +
   "   class " + className +
   "   {" +
   "      public double " + methodName + "(double value1, double value2)" +
   "      {" +
   "         return value1 + value2;" +
```

Listing 731: Zusammensetzen eines Beispiel-Quellcodes

```
"       }" +
"   }" +
"}";
```

Listing 731: Zusammensetzen eines Beispiel-Quellcodes (Forts.)

Je nach Programmiersprache (ich verwende hier natürlich C#) müssen Sie sich nicht um Zeilenumbrüche kümmern, da die Compiler für die Sprachen mit Zeilenende-Zeichen (C#, J#) diese sowieso ignorieren. In dem Beispiel habe ich deswegen auch keine Zeilenumbrüche im Code-String eingesetzt und den Quelltext lediglich so zusammengestellt, dass er für Sie verständlich ist.

Beim Aufbau des Quellcodes haben Sie alle Möglichkeiten, die die Sprache bietet. Sie müssen lediglich beachten, dass Sie die Methode(n) später über Reflection aufrufen und deswegen den Namensraum, die Klasse und den Namen und die Signatur der Methode kennen müssen. Wenn Sie Quellcode ausführen wollen, der vom Anwender eingegeben wird, müssen Sie diese Namen also festlegen. Dem Anwender können Sie in diesem Fall lediglich die Möglichkeit bieten, den Inhalt der Methode(n) anzugeben.

Zum Kompilieren des Quellcodes benötigen Sie eine Instanz der `CodeDomProvider`-Klasse, die Sie über die statische `CreateProvider`-Methode erhalten. Dieser Methode übergeben Sie einen String, der die zu verwendende Sprache angibt. "C#" steht dabei natürlich für C#. Über die `CompileAssemblyFromSource`-Methode der `CodeDomProvider`-Instanz kompilieren Sie den Quellcode. Dabei übergeben Sie am ersten Argument eine Instanz der Klasse `CompilerParameters` und am zweiten den Code.

Die `CompilerParameters`-Instanz verwaltet alle Parameter, die dem Compiler übergeben werden sollen. Die Eigenschaft `GenerateInMemory` legt fest, dass die Assembly im Speicher erzeugt werden soll. Über `GenerateExecutable` legen Sie fest, ob Sie eine ausführbare (Exe-)Datei oder eine Klassenbibliothek kompilieren. `IncludeDebugInformation` bestimmt, ob Debuginformationen mit in die Assembly aufgenommen werden sollen. Die letzte für unser Vorhaben wichtige Eigenschaft ist `ReferencedAssemblys`, über deren `Add`-Methode Sie Referenzen zu allen in der dynamischen Klasse benötigten Assemblys hinzufügen können.

`CompileAssemblyFromSource` gibt ein `CompilerResults`-Objekt zurück, über das Sie später eventuelle Fehler auswerten bzw. die erzeugte Assembly erreichen.

```
// Compiler erzeugen
CodeDomProvider compiler = CodeDomProvider.CreateProvider("C#");

// Input-Parameter für den Compiler definieren
CompilerParameters compilerParams = new CompilerParameters();
compilerParams.GenerateInMemory = true;
compilerParams.GenerateExecutable = false;
compilerParams.IncludeDebugInformation = true;
compilerParams.ReferencedAssemblys.Add("system.dll");
```

Listing 732: Dynamisches Kompilieren von Quellcode in eine In-Memory-Assembly

```
// Assembly erzeugen
CompilerResults compilerResults =
   compiler.CompileAssemblyFromSource(compilerParams, source);
```

Listing 732: Dynamisches Kompilieren von Quellcode in eine In-Memory-Assembly (Forts.)

Nach dem Kompilieren sollten Sie über die `Errors`-Eigenschaft des `CompilerResults`-Objekts überprüfen, ob Fehler aufgetreten sind. Sind keine Fehler aufgetreten, erreichen Sie die erzeugte In-Memory-Assembly über die Eigenschaft `CompiledAssembly`. Der Rest ist reine Reflection: Das Aufrufen einer Methode in einer Assembly.

Dazu holen Sie zunächst über die `GetType`-Methode der Assembly eine `Type`-Instanz der Klasse. Über die `InvokeMember`-Methode dieses Objekts erzeugen Sie dann eine Instanz der Klasse. Dabei müssen Sie einige Konstanten der `BindingFlags`-Aufzählung übergeben. Welche das sind, zeigt Listing 733.

Die Methode rufen Sie schließlich über die `InvokeMember`-Methode des `Type`-Objekts auf, wobei Sie wieder einige Konstanten der `BindingFlags`-Aufzählung übergeben müssen. Neben dem Namen der Methode und den `BindingFlags` übergeben Sie am vierten Argument die Instanz der Klasse und am letzten Argument ein `object`-Array mit den Argumenten der Methode. `InvokeMember` gibt ein `object` zurück, das den Wert speichert, den die Methode zurückgegeben hat.

Sind beim Kompilieren Fehler aufgetreten, können Sie diese aus der `Errors`-Auflistung des `CompilerResults`-Objekts auslesen. Die Eigenschaft `Line` der in dieser Auflistung referenzierten `CompilerError`-Objekte speichert die Zeile, in der der Fehler aufgetreten ist. Aus der Eigenschaft `Column` können Sie Spalten auslesen und `ErrorText` verwaltet die Fehlermeldung.

```
// Auf eventuelle Fehler abfragen
if (compilerResults.Errors.Count == 0)
{
   // Den erzeugten Typ laden
   Type type = compilerResults.CompiledAssembly.GetType(
      assemblyNamespace + "." + className, true);

   // Eine Instanz der Klasse erzeugen
   BindingFlags baseBindingFlags = BindingFlags.DeclaredOnly |
      BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance;
   object classInstance = type.InvokeMember(className,
      baseBindingFlags | BindingFlags.CreateInstance, null, null, null);

   // Die Methode aufrufen
   object[] args = { 10, 11 };
   object result = type.InvokeMember(methodName,
      baseBindingFlags | BindingFlags.InvokeMethod,
      null, classInstance, args);
```


Listing 733: Fehlerauswertung und dynamischer Aufruf der erzeugten Methode

```
   // Das Ergebnis ausgeben
   Console.WriteLine("Das Ergebnis: " + Convert.ToDouble(result));
}
else
{
   // Fehler auswerten
   for (int i = 0; i < compilerResults.Errors.Count; i++)
   {
      Console.WriteLine("Zeile " + compilerResults.Errors[i].Line +
         ", Spalte " + compilerResults.Errors[i].Column + ": " +
         compilerResults.Errors[i].ErrorText);
   }
}
```

Listing 733: Fehlerauswertung und dynamischer Aufruf der erzeugten Methode (Forts.)

Der Rest (also die Anwendung dieser Technik) liegt bei Ihnen ☺.

359 Assemblys dynamisch erzeugen

Das dynamische Erzeugen von Assemblys entspricht bis auf drei kleine Unterschiede dem dynamischen Ausführen von Quellcode. Ich verzichte deswegen in diesem Rezept auf eine nähere Beschreibung und verweise auf das Rezept 358. Die wesentlichen Unterschiede zu diesem Rezept sind, dass Sie die Eigenschaft `GenerateInMemory` des `CompilerParameters`-Objekts auf `false` setzen und in der Eigenschaft `OutputAssembly` den Dateinamen der zu erzeugenden Assembly angeben. Ein weiterer Unterschied ist noch, dass Sie beim dynamischen Erzeugen einer Assembly den Quellcode nicht ausführen müssen.

Das folgende Beispiel erzeugt eine Assembly *Demo.dll* im Ordner der Anwendung. Die in dieser Assembly enthaltene Klasse entspricht der aus dem Rezept 358.

Zum Kompilieren dieses Quellcodes müssen Sie die Namensräume `System`, `System.CodeDom.Compiler`, `System.Reflection` und `System.IO` einbinden.

```
// Den Quellcode zusammenstellen
string assemblyNamespace = "DynamicCode";
string className = "Demo";
string methodName = "Add";
string applicationPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string assemblyName = Path.Combine(applicationPath, "Demo.dll");
string source = "namespace " + assemblyNamespace +
   "{" +
   "   class " + className +
   "   {" +
   "      public double " + methodName + "(double value1, double value2)" +
```

Listing 734: Dynamisches Erzeugen einer Assembly

```
   "      {" +
   "         return value1 + value2;" +
   "      }" +
   "   }" +
   "}";

// Compiler erzeugen
CodeDomProvider compiler = CodeDomProvider.CreateProvider("C#");

// Input-Parameter für den Compiler definieren
CompilerParameters compilerParams = new CompilerParameters();
compilerParams.GenerateInMemory = false;
compilerParams.OutputAssembly = assemblyName;
compilerParams.GenerateExecutable = false;
compilerParams.IncludeDebugInformation = true;
compilerParams.ReferencedAssemblys.Add("system.dll");

// Assembly erzeugen
CompilerResults compilerResults =
   compiler.CompileAssemblyFromSource(compilerParams, source);

// Auf eventuelle Fehler abfragen
if (compilerResults.Errors.Count == 0)
{
   Console.WriteLine("Die Assembly '" + assemblyName +
      "' wurde erfolgreich erzeugt");
}
else
{
   // Fehler auswerten
   for (int i = 0; i < compilerResults.Errors.Count; i++)
   {
      Console.WriteLine("Zeile " + compilerResults.Errors[i].Line +
         ", Spalte " + compilerResults.Errors[i].Column + ": " +
         compilerResults.Errors[i].ErrorText);
   }
}
```

Listing 734: Dynamisches Erzeugen einer Assembly (Forts.)

360 Assemblys dynamisch laden

Das dynamische Laden einer Assembly (um diese per Reflection auswerten oder deren Methoden ausführen zu können) ist prinzipiell sehr einfach. Über `System.Reflection.Assembly.LoadFrom` können Sie eine Assembly in die aktuelle Anwendungsdomäne laden. `LoadFrom` ermöglicht neben dem Ermitteln von Informationen auch das Erzeugen von Typen.

Tipp

Falls Sie ein Add-In- bzw. Plug-In-System für Ihre Anwendung(en) planen, sollten Sie sich die .NET-Framework-Klassen aus dem Namensraum `System.Addin` anschauen, die das Entwickeln von Add-Ins sehr einfach machen. Im MSDN-Magazin finden Sie Artikel, die das Ganze ausführlich beschreiben. Der Artikel »Tiefe Einblicke in die CLR, Erweiterbarkeit von .NET-Anwendungen«, dessen ersten Teil Sie an der Adresse *msdn.microsoft.com/msdnmag/issues/07/02/CLRInsideOut/Default.aspx* finden, führt in das Thema ein. Den zweiten Teil finden Sie an der Adresse *msdn.microsoft.com/msdnmag/issues/07/03/CLRInsideOut/Default.aspx*.

In manchen Fällen ist das dynamische Laden von Assemblys u. U. problematisch, wenn die Assembly z. B. für eine verzögerte Signierung markiert wurde oder aufgrund von Code-Access-Security-Regeln auf dem aktuellen System gar nicht ausgeführt werden darf. Wenn Sie nur Informationen aus der Assembly auslesen wollen (z. B. alle exportierten Klassen), können Sie stattdessen auch die `ReflectionOnlyLoadFrom`-Methode aufrufen. Diese Methode ermöglicht nicht das Erzeugen von Typen, übergeht bei der Verwendung von Reflection aber die Verifizierung eines starken Namens, den CAS-Check und die Prozessor-Architektur-Regeln.

Assemblys, die andere Assemblys referenzieren, können allerdings nur dann verwendet werden, wenn auch die anderen Assemblys in der Anwendungsdomäne geladen sind. Dieses Problem löst die CLR automatisch, wenn die referenzierten Assemblys im Programmordner, im Ordner der zu ladenden Assemblys oder in einem Ordner gespeichert sind, der in der Konfiguration als Assembly-Ordner angegeben ist. In allen anderen (wohl nur sehr selten auftretenden) Fällen können Sie Referenz-Probleme lösen, indem Sie für das Ereignis `AssemblyResolve` (für `Assembly.LoadFrom`) bzw. `ReflectionOnlyAssemblyResolve` (für `Assembly.ReflectionOnlyLoadFrom`) des `AppDomain`-Objekts eine Methode implementieren. In dieser Methode erhalten Sie den Namen der referenzierten Assembly, müssen diese laden und eine Referenz darauf zurückgeben.

Ist die Assembly geladen, können Sie diese über Reflection evaluieren oder die enthaltenen Methoden dynamisch ausführen. Beim Ausführen von Methoden (z. B. für ein Plug-In-System, bei dem der Anwender spezielle Plug-In-Assemblys in einen dafür vorgesehenen Ordner kopieren kann) ist es von Vorteil, wenn die dafür vorgesehenen Klassen eine Schnittstelle implementieren, die auch der Anwendung bekannt ist. So können Sie die über Reflection erhaltene Instanz der Klasse in die Schnittstelle casten und typsicher auf deren Member zugreifen.

Um dem Ganzen ein wenig Praxisnähe zu geben, habe ich für dieses Rezept eine Assembly *Demo-Basis-Assembly.dll* mit einer einfachen Methode implementiert, die den Namen des aktuellen Benutzers zurückgibt:

```
using System;

namespace Addison_Wesley.Codebook.Demos
{
   public class DemoBase
   {
      public static string GetUserName()
```

Listing 735: Basis-Assembly für die Demo-Plug-Ins

```
      {
         return Environment.UserName;
      }

   }
}
```

Listing 735: Basis-Assembly für die Demo-Plug-Ins (Forts.)

Die Assemblys *Demo1.dll* und *Demo2.dll* referenzieren diese Assembly und rufen deren `GetUserName`-Methode auf. Außerdem referenzieren diese Assemblys die Assembly *InterfaceAssembly.dll*, die lediglich eine Schnittstelle verwaltet:

```
namespace Addison_Wesley.Codebook.Demos
{
   public interface IDemo
   {
      void GiveMessage(string message);
   }
}
```

Listing 736: Interface-Assembly

Demo1.dll implementiert die `GiveMessage`-Methode der `IDemo`-Schnittstelle folgendermaßen:

```
using System;

namespace Addison_Wesley.Codebook.Demos
{
   public class Demo: IDemo
   {
      public void GiveMessage(string message)
      {
         Console.WriteLine("Hallo " +  DemoBase.GetUserName()  +
            ". Ich bin Demo1. Hier ist die Nachricht: " + message);
      }
   }
}
```

Listing 737: Implementierung der GiveMessage-Methode in der Plug-In-Assembly Demo1.dll.

Demo2.dll besitzt eine ähnliche Implementierung, nur dass statt »Ich bin Demo1« der Text »Ich bin Demo2« ausgegeben wird.

Das Beispiel-Programm, das die Interface-Assembly ebenfalls referenziert, sucht zunächst nach allen Plug-In-Assemblys im Ordner *Plug-Ins*. Gefundene Plug-Ins (die hier am Namen

erkannt werden) werden über die `LoadFrom`-Methode der `Assembly`-Klasse in die aktuelle Anwendungsdomäne geladen. Über die `GetType`-Methode ermittelt das Programm dann eine `Type`-Instanz für die in der Plug-In-Assembly gespeicherte Klasse `Demo` und erzeugt über deren `InvokeMember`-Methode eine Instanz der Klasse. Diese wird nun in die Schnittstelle `IDemo` gecastet, da die Demo-Klassen diese Schnittstelle ja implementieren. Der Rest ist einfaches Zugreifen auf die einzige Methode dieser Schnittstelle.

Zum Kompilieren des Beispiels müssen Sie die Assembly *Interface-Assembly.dll* referenzieren (die Sie im Beispiel zu diesem Rezept finden) und die Namensräume `System.IO`, `System.Reflection` und `Addison_Wesley.Codebook.Demos` importieren.

```
// Suchen nach Plug-In-Assemblys (deren Name hier mit "Demo" beginnt)
string applicationPath = Path.GetDirectoryName(
   Assembly.GetEntryAssembly().Location);
string plugInPath = Path.Combine(applicationPath, "Plug-Ins");
DirectoryInfo di = new DirectoryInfo(plugInPath);
foreach (FileInfo fi in di.GetFiles("Demo*.dll"))
{
   if (fi.Name != "Demo-Basis-Assembly.dll")
   {
      // Laden dieser Assembly
      Assembly assembly = Assembly.LoadFrom(fi.FullName);

      // Holen einer Instanz der Klasse Demo
      Type classType = assembly.GetType(
        "Addison_Wesley.Codebook.Demos.Demo", true);
      BindingFlags baseBindingFlags = BindingFlags.DeclaredOnly |
         BindingFlags.Public | BindingFlags.NonPublic |
         BindingFlags.Instance;
      object classInstance = classType.InvokeMember("Demo",
         baseBindingFlags | BindingFlags.CreateInstance, null, null, null);

      // Casten in das Interface
      IDemo demo = (IDemo)classInstance;
      // Aufrufen der Methode
      demo.GiveMessage("Das ist ein Test");
   }
}
```

Listing 738: Dynamisches Laden einer Assembly und Ausführen einer Methode

Das Programm hat kein Problem mit der Auflösung der Referenz zu der von *Demo1.dll* und *Demo2.dll* referenzierten Assembly *Demo-Basis-Assembly.dll*, da diese für das Beispiel in demselben Ordner gespeichert ist, wie die dynamisch geladenen Assemblys. Abbildung 207 beweist, dass das Programm funktioniert.

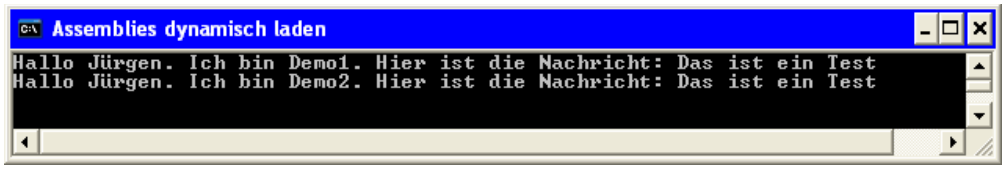


Abbildung 207: Das Beispielprogramm in Aktion

Hinweis

Ein kleines Problem meiner Vorgehensweise ist, dass die geladenen Assemblys nicht wieder entladen werden können. Laden Sie im Verlauf des Programms mehrere Assemblys, die das Programm nur temporär benötigt, kann das zum (Speicher-)Problem werden. Eine Lösung für dieses Problem ist das Erzeugen einer neuen Anwendungsdomäne, in die Sie die Assembly laden. Über `AppDomain.Unload` können Sie diese Anwendungsdomäne komplett aus dem Speicher entfernen. Das Problem dabei ist lediglich, dass Sie in der eigentlichen Anwendung keinen Typen der dynamisch geladenen Assembly und natürlich auch nicht die Assembly selbst referenzieren dürfen. In diesem Fall wird die Assembly nämlich zusätzlich in die Anwendungsdomäne des Programms geladen. Dieses Problem können Sie lösen, indem Sie die Methoden und Eigenschaften der dynamisch geladenen Assembly über Remoting ansprechen. Eine andere, bessere Lösung ist die Verwendung einer schmalen Assembly, die lediglich der Weitergabe der Member-Aufrufe dient. Diese als *Shim* bezeichnete Assembly wird in die temporäre Anwendungsdomäne geladen und kann in der Anwendungsdomäne des Programms referenziert werden. Einen entsprechenden Artikel finden Sie an der (leider etwas unschönen) Adresse *www.devsource.com/article2/0,1759,1790388,00.asp*.

361 Objekte über eine Datenvertrag-Serialisierung serialisieren

Bei der Serialisierung über Datenvertrag-Serialisierer, die ab dem .NET Framework 3.0 verfügbar ist, arbeiten Sie mit einer von zwei möglichen Klassen aus dem Namensraum `System.Runtime.Serialization`:

- `DataContractSerializer`: Koppelt .NET-Typen über einen Datenvertrag *lose* an serialisierte Daten. Beim Deserialisieren kann ein vollkommen anderer Typ verwendet werden als beim Serialisieren. Die einzige Voraussetzung ist, dass dieser Typ den Datenvertrag einhält.
- `NetDataContractSerializer`: Koppelt .NET-Typen über einen Datenvertrag *eng* an serialisierte Daten. Die enge Kopplung wird dadurch erreicht, dass der vollständige serialisierte Typ über die zwei zusätzlichen Attribute im erzeugten XML-Dokument angegeben wird. Die Daten können nur über genau diesen Typ wieder deserialisiert werden (was die Idee eines Datenvertrags eigentlich komplett aufhebt).

Der (Daten-)Vertrag der Datenvertrag-Serialisierer besteht aus:

- einem XML-Namensraum,
- einem XML-Namen für die serialisierten Typen
- und Informationen darüber, welche Felder serialisiert werden und wie diese in XML heißen.

Der Datenvertrag wird bei beiden Datenvertrag-Serialisierern über Attribute in den Klassen oder Strukturen definiert, die die zu serialisierenden Objekte beinhalten. Das Attribut `DataContract` bestimmt zunächst, dass der Typ grundsätzlich serialisierbar ist. Dieses Attribut wird vor den Typnamen angegeben. Über Eigenschaften der Attributklasse können Sie Teilinformationen bestimmen:

- `Name`: Bestimmt den Namen des Datenvertrags. Mit diesem Namen wird das Wurzelelement des XML-Dokuments gekennzeichnet. Wenn Sie den Namen nicht angeben, wird automatisch der Typname verwendet.
- `Namespace`: Bestimmt den XML-Namensraum. Wenn Sie den Namensraum nicht explizit angeben, wird der Namensraum »http://schemas.datacontract.org/2004/07/*.NET-Namensraum*« verwendet. *.NET-Namensraum* entspricht dabei dem .NET-Namensraum des serialisierten Typs.

Die Felder bzw. Eigenschaften des Typs, die serialisiert werden sollen, werden mit dem Attribut `DataMember` gekennzeichnet. Hier können Sie den Datenvertrag wieder über Attribut-Eigenschaften beeinflussen:

- `Name`: Bestimmt den Namen des XML-Elements, das den Wert des Feldes oder der Eigenschaft verwaltet. Geben Sie den Namen nicht an, wird automatisch der Name des Feldes bzw. der Eigenschaft verwendet.
- `IsRequired`: Legt fest, ob das Feld bzw. die Eigenschaft beim Deserialisieren erforderlich ist. Die Voreinstellung ist `true`.
- `EmitDefaultValue`: Bestimmt, ob der Standardwert des Feldes bzw. der Eigenschaft serialisiert werden soll. Die Voreinstellung ist `true`. Wenn Sie `false` angeben, wird das Feld bzw. die Eigenschaft nicht serialisiert, wenn dieses den Standardwert aufweist.
- `Order`: Dieser `int`-Wert bestimmt, an welcher Position das Feld bzw. die Eigenschaft in den serialisierten Daten angelegt wird. Die Voreinstellung -1 bewirkt, dass der Serialisierer die Reihenfolge bestimmt.

Beim Serialisieren sollten Sie zumindest den XML-Namensraum bestimmen, wenn die Daten an andere Systeme übertragen werden sollen. Für den Fall, dass Ihre Typen nicht mit sprechenden Bezeichnern benannt sind (oder mit deutschen, die serialisierten Daten aber international bearbeitet werden sollen), sollten Sie auch die `Name`-Eigenschaft festlegen.

In der folgenden `Person`-Klasse, die bereits gute Bezeichner verwendet, lege ich jedoch nur den Namensraum fest:

```
[DataContract(
   Namespace="http://www.addison-wesley.de/contracts/Serialization-Demo")]
public class Person
{
   [DataMember]
   public string FirstName;
```

Listing 739: Eine Klasse mit der Beschreibung eines Datenvertrags

```
   [DataMember]
   public string LastName;

   /* Dieses Feld soll nicht serialisiert werden */
   public string EMailAddress;
}
```

Listing 739: Eine Klasse mit der Beschreibung eines Datenvertrags (Forts.)

Dass ich im Beispiel das `EMailAddress`-Feld nicht mit dem `DataMember`-Attribut gekennzeichnet habe, bewirkt, dass dieses Feld nicht serialisiert wird. Dies soll nur demonstrieren, dass Sie (natürlich) auch Felder von der Serialisierung ausschließen können. Sinn würde das zum Beispiel machen, wenn Sie die Personen-Daten über ein Netzwerk übertragen, die E-Mail-Adresse der Person aber geheim bleiben soll. Wenn Sie Daten lokal serialisieren und wieder deserialisieren, werden Sie wahrscheinlich aber alle Felder des zu serialisierenden Typs mit dem `DataMember`-Attribut kennzeichnen, um bei der Serialisierung keine Informationen zu verlieren.

Typen, die über den speziellen Attribut-basierten Datenvertrag vorbereitet sind, können dann z. B. in WCF für die Übertragung zwischen verschiedenen Systemen verwendet werden. Sie können aber auch die Datenvertrag-Serialisierungs-Klassen verwenden, um Instanzen dieser Typen selbst zu serialisieren und wieder zu deserialisieren.

Die Serialisierung

Zur Serialisierung erzeugen Sie zunächst eine Instanz der Klasse `DataContractSerializer` oder `NetDataContractSerializer`. Dem Konstruktor von `DataContractSerializer` müssen Sie den Typ übergeben, den Sie serialisieren oder in den Sie deserialisieren wollen. Diese Übergabe ist eher für die Deserialisierung wichtig, da Sie mit `DataContractSerializer` auch in anderen Typen deserialisieren können. Da `NetDataContractSerializer` die Typinformationen in den XML-Daten verwaltet, benötigt diese Klasse keine Typangabe.

Über die Methode `WriteObject` können Sie dann eine Instanz dieses Typs serialisieren. Dieser Methode übergeben Sie einen `Stream`, ein `XmlDictionaryWriter`- oder ein `XmlWriter`-Objekt. Damit können Sie die Daten z. B. direkt (formatiert) in eine XML-Datei schreiben:

```
// Eine Instanz der Person-Klasse erzeugen
Person person = new Person(){
   FirstName = "Zaphod", LastName = "Beeblebrox",
   EMailAddress = "Zaphod@Galaxy.com"};

// Die Instanz über einen DataContractSerializer serialisieren
DataContractSerializer dataContractSerializer =
   new DataContractSerializer(typeof(Person));
using (Stream stream = new FileStream("C:\\Person1.xml",
   FileMode.Create))
{
   using (XmlWriter xmlWriter = XmlWriter.Create(stream,
```

Listing 740: Serialisieren über die Datenvertrag-Serialisierer

```
      new XmlWriterSettings() { Indent = true }))
   {
      dataContractSerializer.WriteObject(xmlWriter, person);
   }
}

// Die Instanz über einen NetDataContractSerializer serialisieren
NetDataContractSerializer netDataContractSerializer =
   new NetDataContractSerializer();
using (Stream stream = new FileStream("C:\\Person2.xml",
   FileMode.Create))
{
   using (XmlWriter xmlWriter = XmlWriter.Create(stream,
      new XmlWriterSettings() { Indent = true }))
   {
      netDataContractSerializer.WriteObject(xmlWriter, person);
   }
}
```

Listing 740: Serialisieren über die Datenvertrag-Serialisierer (Forts.)

Die Deserialisierung

Zum Deserialisieren eines serialisierten Objekts verwenden Sie die `ReadObject`-Methode. Ein mit dem `NetDataContractSerializer` serialisiertes Objekt können Sie nur in genau denselben Typen deserialisieren:

```
NetDataContractSerializer netDataContractSerializer =
   new NetDataContractSerializer();
using (Stream stream = new FileStream("C:\\Person2.xml", FileMode.Open,
   FileAccess.Read))
{
    Person p = (Person)netDataContractSerializer.ReadObject(stream);
    Console.WriteLine(p.FirstName + " " + p.LastName + " - " +
       (p.EMailAddress != null ? p.EMailAddress :
       "Keine E-Mail-Adresse"));
}
```

Listing 741: Deserialisieren über einen NetDataContractSerializer

Eine `DataContractSerializer`-Instanz erlaubt hingegen auch das Deserialisieren in einen beliebigen Typ. Dieser muss lediglich den Datenvertrag einhalten.

Wenn Sie sich jetzt fragen, wofür Sie die einzelnen Möglichkeiten brauchen:

- Verwenden Sie einen `DataContractSerializer`, wenn die Daten zwischen verschiedenen Systemen ausgetauscht werden, die nicht beide Zugriff auf dieselbe Assembly besitzen, die den serialisierten Typ enthält. `DataContractSerializer` eignet sich also hervorragend zum Austausch von Objekten (bzw. deren Daten) über ein Netzwerk.

- Verwenden Sie einen `NetDataContractSerializer`, wenn sowohl der Sender als auch der Empfänger die Assembly referenzieren können, die den serialisierten Typ enthält. `NetDataContractSerializer` eignet sich also im Wesentlichen zum einfachen Verwalten von Objekten in einer Anwendung oder zum Austausch von Objektdaten zwischen mehreren .NET-Anwendungen, die dieselbe Assembly referenzieren.

Objekte binär serialisieren

Über die Klassen `XmlDictionaryWriter` und `XmlDictionaryReader` können Sie Objekte auch binär serialisieren und deserialisieren:

```
Person person = new Person() ...

// Die Instanz über einen DataContractSerializer serialisieren
DataContractSerializer dataContractSerializer =
   new DataContractSerializer(typeof(Person));
using (Stream stream = new FileStream("C:\\Person.dat", FileMode.Create))
{
   using (XmlDictionaryWriter writer =
      XmlDictionaryWriter.CreateBinaryWriter(stream))
   {
      dataContractSerializer.WriteObject(writer, person);
   }
}

// Die Datei über den DataContractSerializer deserialisieren
using (Stream stream = new FileStream("C:\\Person.dat", FileMode.Open,
   FileAccess.Read))
{
   using (XmlDictionaryReader reader =
      XmlDictionaryReader.CreateBinaryReader(stream,
      XmlDictionaryReaderQuotas.Max))
   {
      Person p = (Person)dataContractSerializer.ReadObject(reader);
      ...
   }
}
```

Listing 742: Binäres Serialisieren und Deserialisieren mit einem Datenvertrag-Serialisierer

Die Größe der erzeugten Daten ist natürlich kleiner als bei der Verwendung von XML-Daten. Der Nachteil ist, dass die Daten nicht von einem Mensch bearbeitet werden können. Außerdem kann es bei der Übertragung über ein Netzwerk zu Problemen mit Firewalls kommen, die binäre Daten nicht durchlassen.

362 Objekte binär serialisieren und deserialisieren

Wenn der Zustand eines Objekts zwischen zwei Programmaufrufen bestehen bleiben soll, wenn Sie ein Objekt von einem Programm zu einem anderen Programm oder System versenden wollen oder wenn das Objekt in einer Datenbank gespeichert werden soll, müssen Sie das Objekt serialisieren. Dabei werden die Daten des Objekts in einen Stream geschrieben, den Sie dann beliebig weiterverarbeiten können.

Zum Serialisieren und Deserialisieren können Sie die Klasse `BinaryFormatter` aus dem Namensraum `System.Runtime.Serialization.Formatters.Binary` verwenden. Die Methode `Serialize` serialisiert ein Objekt in einen binären Stream, die Methode `Deserialize` deserialisiert einen binären Stream in ein Objekt. Dabei werden alle Daten des Objekts – auch die privaten – in den Stream geschrieben. Im Rezept 363 finden Sie übrigens eine andere Möglichkeit, nämlich das Serialisieren nach XML.

Die einzige Voraussetzung für das Serialisieren ist, dass die Klasse des Objekts mit dem Attribut `Serializable` gekennzeichnet ist:

```
[Serializable]
public class Person
{
   public string FirstName;
   public string LastName;
   public DateTime BirthDate;
}
```

Listing 743: Klasse für serialisierbare Objekte

Objekte einer solchen Klasse können Sie dann sehr einfach serialisieren und deserialisieren. Die Methode `SerializeToFile` in Listing 744 zeigt, wie Sie ein Objekt in eine Datei serialisieren. Zum Kompilieren dieser Methode müssen Sie die Namensräume `System.IO` und `System.Runtime.Serialization.Formatters.Binary` importieren.

```
public static void SerializeToFile(object obj, string filename)
{
   // FileStream für die Datei erzeugen
   using (FileStream fileStream = new FileStream(filename,
      FileMode.Create, FileAccess.Write))
   {
      // Objekt serialisieren
      BinaryFormatter binaryFormatter = new BinaryFormatter();
      binaryFormatter.Serialize(fileStream, obj);
   }
}
```

Listing 744: Methode zum Serialisieren eines Objekts in eine Datei

Das Deserialisieren eines Objekts aus einer Datei ist ebenfalls sehr einfach:

```
public static object DeserializeFromFile(string filename)
{
   // FileStream für die Datei erzeugen
   using (FileStream fileStream = new FileStream(filename,
      FileMode.Open, FileAccess.Read))
   {
      // Objekt deserialisieren
      BinaryFormatter binaryFormatter = new BinaryFormatter();
      return binaryFormatter.Deserialize(fileStream);
   }
}
```

Listing 745: Deserialisieren eines Objekts aus einer Datei

Beide Methoden behandeln Ausnahmen nicht um diese direkt an den Aufrufer weiterzugeben.

Um Objekte in einer Datenbank speichern zu können, müssen Sie diese in ein Byte-Array serialisieren. Dieses Array können Sie dann in Datenbankfelder vom Typ BLOB (Binary Large Object) oder ähnliche Felder schreiben und daraus auslesen.

Zum Serialisieren in ein Byte-Array und zum Deserialisieren aus einem solchen Array können Sie ein `MemoryStream`-Objekt (aus dem Namensraum `System.IO`) verwenden. Die Methoden `SerializeToByteArray` und `DeserializeFromByteArray` in Listing 746 zeigen, wie Sie dies programmieren können. Beide Methoden erfordern den Import der Namensräume `System.IO` und `System.Runtime.Serialization.Formatters.Binary`.

```
/* Serialisiert ein Objekt in ein Byte-Array */
public static byte[] SerializeToByteArray(object obj)
{
   // MemoryStream erzeugen
   using (MemoryStream memoryStream = new MemoryStream())
   {
      // Objekt serialisieren
      BinaryFormatter binaryFormatter = new BinaryFormatter();
      binaryFormatter.Serialize(memoryStream, obj);

      return memoryStream.ToArray();
   }
}

/* Deserialisiert ein Objekt aus einem Byte-Array */
public static object DeserializeFromByteArray(byte[] data)
{
```

Listing 746: Methoden zum Serialisieren und Deserialisieren eines Objekts in und aus einem Byte-Array

```
   // MemoryStream erzeugen
   using (MemoryStream memoryStream = new MemoryStream(data))
   {
      // Objekt deserialisieren
      BinaryFormatter binaryFormatter = new BinaryFormatter();
      return binaryFormatter.Deserialize(memoryStream);
   }
}
```

Listing 746: Methoden zum Serialisieren und Deserialisieren eines Objekts in und aus einem Byte-Array (Forts.)

Das folgende Beispiel zeigt, wie Sie mit Hilfe der Methoden `SerializeToFile` und `DeserializeFromFile` ein `Person`-Objekt in eine Datei serialisieren und wieder daraus deserialisieren. Beim Deserialisieren ist eine Typumwandlung notwendig, da `DeserializeFromFile` eine `object`-Referenz zurückgibt. Das Beispiel benötigt aufgrund der Verwendung der `Application`-Klasse die Referenzierung der Assembly *System.Windows.Forms.dll* und den Import des Namensraums `System.Windows.Forms`. Daneben müssen Sie die Namensräume `System` und `System.IO` importieren.

```
// Person-Objekt erzeugen
Person person = new Person();
person.FirstName = "Zaphod";
person.LastName = "Beeblebox";
person.BirthDate = new DateTime(1900, 1, 1);

// Objekt in eine Datei serialisieren
string fileName = Path.Combine(Application.StartupPath, "Person.dat");
SerializeToFile(person, fileName);

// Objekt aus einer Datei deserialisieren
person = (Person)DeserializeFromFile(fileName);

Console.WriteLine("{0}\r\n{1}\r\n{2}", person.FirstName, person.LastName,
   person.BirthDate.ToShortDateString());
```

Listing 747: Serialisieren eines Objekts in eine Datei

363 Objekte nach XML serialisieren und von XML deserialisieren

Die Datenvertrags-Serialisierung (Rezept 361) bietet sehr gute Möglichkeiten, Objekte nach XML zu serialisieren. Wenn Sie allerdings keinen Datenvertrag definieren wollen oder können (weil Sie die Klassen z.B. nicht im Quellcode vorliegen haben), können Sie zum Serialisieren nach XML auch die Klasse `XmlSerializer` aus dem Namensraum `System.Xml.Serialization` verwenden. Über die Methode `Serialize` können Sie ein Objekt in einen `Stream` oder ein `XmlWriter`- oder `TextWriter`-Objekt serialisieren, `Deserialize` deserialisiert die Daten eines `Stream`-, `Xml-`

Reader- oder TextReader-Objekts in ein Objekt. Mit diesen Methoden können Sie ein Objekt sehr einfach in eine Datei oder einen XML-String schreiben.

Hinweis

Anders als bei der binären Serialisation werden dabei nur die öffentlichen Daten in das XML-Dokument geschrieben. Private und geschützte Daten werden nicht berücksichtigt. Für Eigenschaften, die mit set-/get-Methoden auf private Daten zugreifen, ist das kein Problem: Diese werden korrekt ausgewertet. Problematisch wird die XML-Serialisation nur dann, wenn private Daten ausschließlich über den Konstruktor oder über normale Methoden geschrieben werden und keine oder nur Eigenschaften zum Lesen dieser Daten zur Verfügung stehen. In diesem Fall wird aber keine Ausnahme erzeugt. Die Daten werden einfach nicht in das XML-Dokument geschrieben.

Eine weitere Einschränkung ist, dass das zu serialisierende Objekt einen (parameterlosen) Standardkonstruktor zur Verfügung stellen muss. Objekte, die keinen solchen besitzen (wie z. B. Font-Instanzen), können Sie leider nicht in XML serialisieren. Dazu finden Sie aber eine Lösung im Rezept 364.

Die Methode SerializeToXmlFile in Listing 748 serialisiert ein Objekt in eine XML-Datei. Da bei XML-Dateien deren Codierung wichtig ist, wird diese neben dem Objekt und dem Dateinamen am dritten Argument übergeben. Bei der Erzeugung einer XmlSerializer-Instanz wird der Typ des Objekts dem Konstruktor übergeben. SerializeToXmlFile serialisiert das übergebene Objekt dann über eine StreamWriter-Instanz. Auf diese Weise kann die Codierung mit übergeben werden.

Zum Kompilieren dieser Methode (und der folgenden) müssen Sie die Namensräume System, System.IO, System.Text und System.Xml.Serialization importieren.

```
public static void SerializeToXmlFile(object obj, string filename,
   Encoding encoding)
{
   // XmlSerializer für den Typ des Objekts erzeugen
   XmlSerializer serializer = new XmlSerializer(obj.GetType());

   // Objekt über ein StreamWriter-Objekt serialisieren
   using (StreamWriter streamWriter = new StreamWriter(filename, false,
      encoding))
   {
      serializer.Serialize(streamWriter, obj);
   }
}
```

Listing 748: Methode zum Serialisieren eines Objekts in eine XML-Datei

Die Methode SerializeToXmlString in Listing 749 serialisiert ein Objekt auf eine ähnliche Weise in einen String, allerdings vereinfacht über ein StringWriter-Objekt.

```
public static string SerializeToXmlString(object obj)
{
   // XmlSerializer für den Typ des Objekts erzeugen
   XmlSerializer serializer = new XmlSerializer(obj.GetType());

   // Objekt über einen StringWriter serialisieren
   using (StringWriter stringWriter = new StringWriter())
   {
      serializer.Serialize(stringWriter, obj);

      // Das Ergebnis zurückgeben
      return stringWriter.ToString();
   }
}
```

Listing 749: Methode zum Serialisieren eines Objekts in einen XML-String

Das Deserialisieren aus einer XML-Datei übernimmt die Methode `DeserializeFromXmlFile` in Listing 750. Diese Methode ruft zum Deserialisieren die `Deserialize`-Methode eines `XmlSerializer`-Objekts auf. Da dabei wieder der Typ des Objekts übergeben werden muss, erwartet `DeserializeFromXmlFile` diesen am Argument `objectType`. Als Basis für die Deserialisation wird ein `StreamReader`-Objekt verwendet, da ein solches mit einem `Encoding`-Objekt initialisiert werden kann und damit die definierte Codierung berücksichtigt.

```
public static object DeserializeFromXmlFile(string filename,
   Type objectType, Encoding encoding)
{
   // XmlSerializer für den Typ des Objekts erzeugen
   XmlSerializer serializer = new XmlSerializer(objectType);

   // Objekt über ein StreamReader-Objekt serialisieren
   using (StreamReader streamReader = new StreamReader(filename, encoding))
   {
      return serializer.Deserialize(streamReader);
   }
}
```

Listing 750: Methode zum Deserialisieren eines Objekts aus einer XML-Datei

Die Methode `DeserializeFromXmlString` in Listing 751 deserialisiert auf eine ähnliche Weise einen XML-String über ein `StringReader`-Objekt, das mit dem übergebenen String initialisiert wird.

```
public static object DeserializeFromXmlString(string xmlString,
   Type objectType)
```

Listing 751: Methode zum Deserialisieren eines XML-Strings

```
{
   // XmlSerializer für den Typ des Objekts erzeugen
   XmlSerializer serializer = new XmlSerializer(objectType);

   // Objekt über ein StreamReader-Objekt deserialisieren
   using (StringReader stringReader = new StringReader(xmlString))
   {
      return serializer.Deserialize(stringReader);
   }
}
```

Listing 751: Methode zum Deserialisieren eines XML-Strings (Forts.)

Listing 752 zeigt ein `Person`-Objekt (aus der Klasse `Person`, die in Rezept 362 beschrieben wird), das über `SerializeToXmlFile` serialisiert wurde.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Person xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
  <FirstName>Zaphod</FirstName>
  <LastName>Beeblebrox</LastName>
  <BirthDate>1900-01-01T00:00:00.0000000+01:00</BirthDate>
</Person>
```

Listing 752: Ein über SerializeToXmlFile serialisiertes Person-Objekt

364 Font- und andere (serialisierbare) Objekte in einen String serialisieren

In vielen Anwendungen ist es notwendig, eine Schriftart, die vom Anwender ausgewählt werden kann, in der Anwendungs-Konfiguration oder in einer Datenbank zu verwalten, um diese beim nächsten Programmstart wieder auslesen zu können. `Typeface`- bzw. `Font`-Objekte lassen sich aber leider nicht über eine Instanz der Klasse `XmlSerializer` nach XML serialisieren, weil beide Klassen keinen öffentlichen Standardkonstruktor besitzen.

`Font`-Instanzen können Sie allerdings über ein `BinaryFormatter`-Objekt (siehe Rezept 362) in ein binäres und über ein `SoapFormatter`-Objekt in das SOAP-Format serialisieren. `Typeface`-Instanzen können leider nicht serialisiert werden, da die `Typeface`-Klasse nicht als serialisierbar gekennzeichnet ist.

Die Methoden in Listing 753 und Listing 754 zeigen, wie Sie `Font`-Instanzen in Strings serialisieren und aus Strings deserialisieren können. Zum Kompilieren dieser Methoden müssen Sie die Namensräume `System`, `System.Drawing`, `System.IO`, `System.Runtime.Serialization.Formatters.Binary`, `System.Runtime.Serialization.Formatters.Soap` und `System.Text` importieren. Zusätzlich dazu müssen Sie die Assembly *System.Runtime.Serialization.Formatters.Soap.dll* referenzieren.

```
/* Serialisiert ein Font-Objekt in einen SOAP-String */
public static string SerializeFontViaSoap(Font font)
{
   // SoapFormatter und MemoryStream erzeugen
   SoapFormatter soapFormatter = new SoapFormatter();
   using (MemoryStream memoryStream = new MemoryStream())
   {
      // Font-Objekt in den MemoryStream serialisieren
      soapFormatter.Serialize(memoryStream, font);

      // MemoryStream in ein Byte-Array schreiben
      memoryStream.Position = 0;
      byte[] buffer = new byte[memoryStream.Length];
      memoryStream.Read(buffer, 0, buffer.Length);
      memoryStream.Close();

      // Byte-Array in einen UTF8-String konvertieren
      return Encoding.UTF8.GetString(buffer);
   }
}

/* Deserialisiert ein Font-Objekt aus einem SOAP-String */
public static Font DeserializeFontViaSoap(string fontString)
{
   // Den übergebenen SOAP-String über ein Byte-Array in einen
   // MemoryStream schreiben
   byte[] buffer = Encoding.UTF8.GetBytes(fontString);
   using (MemoryStream memoryStream = new MemoryStream(buffer))
   {
      // Den Stream deserialisieren und ein daraus erzeugtes Font-Objekt
      // zurückgeben
      SoapFormatter sf = new SoapFormatter();
      Font font = (Font)sf.Deserialize(memoryStream);
      memoryStream.Close();
      return font;
   }
}
```

Listing 753: Methoden zum Serialisieren/Deserialisieren eines Font-Objekts in bzw. aus einem SOAP-String

Eine andere Möglichkeit, die etwas kleinere Strings erzeugt, ist die binäre Serialisierung über ein `BinaryFormatter`-Objekt. Die so erzeugten Byte-Daten können Sie über `Convert.ToBase64String` in einen Base64-String umwandeln. Beim Deserialisieren konvertieren Sie den Base64-String über `Convert.FromBase64String` in ein Byte-Array, schreiben dies in einen `MemoryStream` und deserialisieren diesen:

```
/* Serialisiert ein Font-Objekt in einen String, der die binäre Darstellung
   des Font-Objekts in BASE64-Form enthält */
public static string SerializeFontViaBinarySerialization(Font font)
{
   // Das Objekt in einen MemoryStream serialisieren
   BinaryFormatter sf = new BinaryFormatter();
   using (MemoryStream memoryStream = new MemoryStream())
   {
      sf.Serialize(memoryStream, font);

      // Den MemoryStream in ein Byte-Array schreiben und dieses in einen
      // Base64-String umgewandelt zurückgeben
      memoryStream.Position = 0;
      byte[] buffer = new byte[memoryStream.Length];
      memoryStream.Read(buffer, 0, buffer.Length);
      memoryStream.Close();
      return Convert.ToBase64String(buffer, 0, buffer.Length);
   }
}

/* Deserialisiert ein Font-Objekt aus einem String,
   der die binäre Darstellung des */
public static Font DeserializeFontViaBinarySerialization(string fontString)
{
   // Den übergebenen Base64-String über ein Byte-Array in einen
   // MemoryStream schreiben
   byte[] buffer = Convert.FromBase64String(fontString);
   using (MemoryStream memoryStream = new MemoryStream(buffer))
   {
      // Den Stream deserialisieren und ein daraus erzeugtes Font-Objekt
      // zurückgeben
      BinaryFormatter sf = new BinaryFormatter();
      Font font = (Font)sf.Deserialize(memoryStream);
      memoryStream.Close();
      return font;
   }
}
```

Listing 754: Methoden zum Serialisieren und Deserialisieren von Font-Objekten über einen Base64-String

Threading und asynchroner Methodenaufruf

365 In einem Thread sicher auf Steuerelemente zugreifen

Wenn Sie in einem Arbeitsthread auf ein Steuerelement zugreifen wollen, sollten Sie dies nicht direkt in dessen Eigenschaften schreiben, Eigenschaften lesen oder Methoden aufrufen. Beim Zugriff auf ein Steuerelement gilt beim Multithreading unter Windows nämlich der Grundsatz, dass nur der Thread, der ein Steuerelement erzeugt hat, auf dieses zugreifen darf. Im Normalfall ist dies der User-Interface(UI)-Thread (bzw. der Hauptthread der Anwendung). Greift ein Arbeitsthread auf ein Steuerelement zu, so kann dies zu Ausnahmen oder zu anderen Problemen wie dem Nicht-mehr-Zeichnen von Teilen des Steuerelements führen. Falls Ausnahmen auftreten, kann es sein, dass diese an Stellen auftreten, die eigentlich gar nichts mit dem Thread zu tun haben, und es kann sein, dass die Art der Ausnahme nicht mit der Programmierung in Verbindung gebracht werden kann.

Bei der Verwendung eines selbst entwickelten Steuerelements aus einem Thread heraus wurde in einem meiner Projekte z. B. eine Scrollbar nicht mehr gezeichnet und es traten `NullReferenceExceptions` in System-Assemblys (!) auf. Solche Probleme sind eigentlich nicht zu debuggen, da sie nur sporadisch auftreten und beim Debuggen selbst meist nicht nachvollziehbar sind.

Auch wenn dieses Problem in (meist sehr einfachen) Testanwendungen nicht aufzutreten scheint: In größeren Anwendungen ist die Wahrscheinlichkeit wesentlich höher, dass es Probleme gibt. Leider treten die Fehler oft erst beim Kunden und dann zu allem Unglück erst nach einiger Zeit auf. In diesem Fall: Viel Spaß bei der Fehlersuche ☺.

Der Grund für diese Probleme liegt in der Nachrichtenverarbeitung unter modernen Windows-Versionen. Hier besitzt jeder Thread eine Nachrichtenwarteschlange, in die die Nachrichten für die Fenster geschrieben werden, die im Thread ausgeführt werden. Probleme entstehen wohl dann, wenn die Aktualisierung eines Steuerelements in einem Arbeitsthread dazu führt, dass das Steuerelement Nachrichten an das Fenster zurücksenden will, auf dem es angelegt wurde. Da die Aktualisierung aber nicht im UI-Thread ausgeführt wird, landen die Nachrichten im falschen Thread.

Als Hilfestellung generiert Visual Studio beim Zugriff auf ein Steuerelement aus einem anderen Thread heraus eine `InvalidOperationException` mit der Meldung »Ungültiger threadübergreifender Vorgang ...«, wenn Sie ein Projekt im Debug-Modus ausführen. Diese Ausnahme wird aber nicht in der kompilierten Anwendung generiert. Sie sollten also den Zugriff auf Steuerelemente und Fenster sehr sorgfältig programmieren oder Ihre Anwendungen immer sehr sorgfältig komplett testen (auch nach Änderungen, was sehr aufwändig sein kann).

Wenn Sie sicher in einem Arbeitsthread auf ein Steuerelement zugreifen wollen, können Sie unter .NET 3.5 drei Wege gehen. Der erste Weg ist die Verwendung einer Methode, deren Aufruf in den Thread umgeleitet wird, der das Steuerelement bzw. Fenster erzeugt hat. Dieser Weg wird hier beschrieben. Der zweite mögliche Weg ist die Verwendung der Komponente `BackgroundWorker`. Diese Komponente beschreibe ich in Rezept 366. Der dritte Weg ist die Imple-

mentierung einer Klasse nach dem ereignisbasierten asynchronen Entwurfsmuster, das ich in Rezept 368 (vereinfacht) erläutere.

Umleitung eines Methodenaufrufs in WPF

In einer WPF-Anwendung leiten Sie Methodenaufrufe über die `Invoke`-Methode des `Dispatcher`-Objekts um, das die `Dispatcher`-Eigenschaft eines Steuerelements oder Fensters liefert. Prinzipiell können Sie immer die `Dispatcher`-Eigenschaft des jeweiligen Fensters verwenden, da dessen Steuerelemente immer auch in demselben Thread laufen (sollten). Möglicherweise bringt aber die Verwendung der `Dispatcher`-Eigenschaft des Objekts, das Sie aktualisieren oder abfragen wollen, einen Performance-Vorteil.

`Invoke` existiert in mehreren Überladungen. Der von mir verwendeten Variante übergeben Sie am ersten Argument einen Wert der `DispatcherPriority`-Aufzählung (aus dem Namensraum `System.Windows.Threading`), die die Priorität des Aufrufs bestimmt. Am zweiten Argument übergeben Sie einen Delegaten mit der auszuführenden Methode. Um das Ganze einfach zu halten, können Sie hier eine Instanz einer der passenden `Action`- oder `Func`-Varianten verwenden und dieser einen Lambda-Ausdruck übergeben. An den folgenden Argumenten übergeben Sie die Argumente, die der Methode übergeben werden sollen. Falls Sie einen `Func`-Delegaten verwenden (die Methode also einen Wert zurückgibt), erhalten Sie den Rückgabewert über den `Object`-Typ, den `Invoke` zurückgibt.

Um zu überprüfen, ob der (zeitintensive) Aufruf von `Invoke` überhaupt notwendig ist, rufen Sie die `CheckAccess`-Methode auf, die `true` zurückgibt, wenn der direkte Zugriff möglich ist.

Das folgende Beispiel zeigt, wie Sie dies programmieren. Es enthält eine Methode, die in einem Thread ausgeführt werden soll. Diese Methode simuliert eine Bearbeitung in einer Schleife. Innerhalb der Schleife wird ein `Label` mit einer Information aktualisiert. `Invoke` wird dazu ein Delegat vom Typ `Action<Label, string>` übergeben. Der Lambda-Ausdruck für diesen Delegaten erhält eine Referenz auf das zu aktualisierende `Label` und den neuen Text übergeben.

Das Programm läuft in einem Fenster, das ein `Label` enthält, das *resultLabel* heißt. Es erfordert den Import der Namensräume `System`, `System.Threading` und `System.Windows.Forms`. Die Methode muss natürlich über eine `Thread`-Instanz gestartet werden.

```
private void DemoThreadMethod()
{
   // Schleife als Demo für die Aktualisierung
   for (int i = 0; i < 100; i++)
   {
      // Sicher auf das Steuerelement zugreifen
      if (this.resultLabel.Dispatcher.CheckAccess())
      {
         // Der Zugriff erfolgt in demselben Thread
         this.resultLabel.Content = i.ToString();
      }
      else
```

Listing 755: Beispiel für den sicheren Zugriff auf ein Steuerelement in einem Thread in einer WPF-Anwendung über Dispatcher.Invoke

```
      {
         // Der Zugriff erfolgt nicht in demselben Thread
         this.resultLabel.Dispatcher.Invoke(
            System.Windows.Threading.DispatcherPriority.Normal,
            new Action<Label, string>(
            (label, content) => label.Content = content),
            this.resultLabel, i.ToString());
      }

      // Kleine Demo-Pause
      Thread.Sleep(50);
   }
}
```

Listing 755: Beispiel für den sicheren Zugriff auf ein Steuerelement in einem Thread in einer WPF-Anwendung über Dispatcher.Invoke (Forts.)

Umleitung eines Methodenaufrufs in Windows.Forms

In einer `Windows.Forms`-Anwendung rufen Sie zur Umleitung eines Methodenaufrufs in den Thread, der ein Steuerelement oder Formular erzeugt hat, dessen `Invoke`-Methode auf. Zur Verwendung dieser Methode benötigen Sie einen Delegaten und eine (anonyme) Methode, die auf das Steuerelement bzw. Formular zugreift. Der Delegat muss die Signatur der Methode besitzen. Am einfachsten verwenden Sie für den Delegaten eine der Überladungen des `Action`-Delegaten und für die Methode einen dazu passenden Lambda-Ausdruck.

`Invoke` übergeben Sie am ersten Argument eine Instanz des Delegaten, der mit der Zugriffsmethode initialisiert wird. Am zweiten Argument übergeben Sie ein `Object`-Array, das mit den Argumenten gefüllt wird, die der Methode übergeben werden sollen. Falls die Methode, die auf das Steuerelement/Formular zugreift, etwas zurückgibt, erhalten Sie den resultierenden Wert über die `Object`-Rückgabe der `Invoke`-Methode.

Über die Eigenschaft `InvokeRequired` des Steuerelements bzw. Formulars können Sie abfragen, ob der (nicht sehr performante) Aufruf der `Invoke`-Methode überhaupt notwendig ist. Nicht notwendig wäre der Aufruf, wenn die Methode, in der Sie auf das Steuerelement zugreifen, in dem Thread ausgeführt wird, der das Steuerelement/Formular erzeugt hat.

Das folgende Beispiel zeigt, wie Sie dies programmieren. Es entspricht im Wesentlichen dem WPF-Beispiel. Das Programm läuft in einem Formular, das ein Label enthält, das *resultLabel* heißt. Es erfordert den Import der Namensräume `System`, `System.Threading` und `System.Windows.Forms`. Die Methode muss natürlich über eine `Thread`-Instanz gestartet werden.

```
private void DemoThreadMethod()
{
   // Schleife als Demo für die Aktualisierung
   for (int i = 0; i < 100; i++)
```

Listing 756: Beispiel für den sicheren Zugriff auf ein Steuerelement in einem Thread in einer Windows.Forms-Anwendung über Invoke

```
   {
      // Sicher auf das Steuerelement zugreifen
      if (this.resultLabel.InvokeRequired)
      {
         // Der Zugriff erfolgt nicht in demselben Thread
         this.resultLabel.Invoke(new Action<Label, string>(
            (label, text) => label.Text = text),
            new object[] { this.resultLabel, i.ToString() });
      }
      else
      {
         // Der Zugriff erfolgt in demselben Thread
         this.resultLabel.Text = i.ToString();
      }

      // Kleine Demo-Pause
      Thread.Sleep(50);
   }
}
```

Listing 756: Beispiel für den sicheren Zugriff auf ein Steuerelement in einem Thread in einer Windows.Forms-Anwendung über Invoke (Forts.)

366 Easy-Threading mit der BackgroundWorker-Komponente

Wie Sie in Rezept 365 erfahren haben, ist ein wichtiger Grundsatz beim Threading, dass nur der Thread, der ein Steuerelement oder Fenster erzeugt hat, auf dieses zugreifen darf. Über eine `BackgroundWorker`-Komponente können Sie diesen Grundsatz in einer einfachen Form sicherstellen. Diese Komponente führt eine Methode in einem Thread aus, ermöglicht die Übergabe von Argumenten an die Thread-Methode, ruft während der Ausführung das Ereignis `ProgressChanged` auf (wenn Sie dies wollen) und ruft am Ende das Ereignis `RunWorkerCompleted` auf. Das Ereignis für den Fortschritt triggern Sie innerhalb der Thread-Methode selbst, wobei Sie einen Prozentwert und ein beliebiges Objekt übergeben können. Innerhalb der `ProgressChanged`-Ereignismethode können Sie gefahrlos auf alle Steuerelemente zugreifen, da diese Methode automatisch in dem Thread aufgerufen wird, der die `BackgroundWorker`-Instanz erzeugt hat (und das ist ja normalerweise auch der Thread, der die anderen Steuerelemente erzeugt).

Zur Programmierung ziehen Sie einfach eine Instanz der `BackgroundWorker`-Komponente auf ein Formular. Falls Sie keine Entwicklungsumgebung besitzen, die Ihnen das ermöglicht, müssen Sie die Instanz natürlich von Hand erzeugen.

Als Beispiel verwende ich eine einfache `Windows.Forms`-Anwendung, die in einem Thread das Ergebnis der Addition aller Zahlen von 1 bis zu einer angegebenen Zahl berechnet. Der Thread kann über den START-Schalter gestartet und über den ABBRECHEN-Schalter abgebrochen werden.

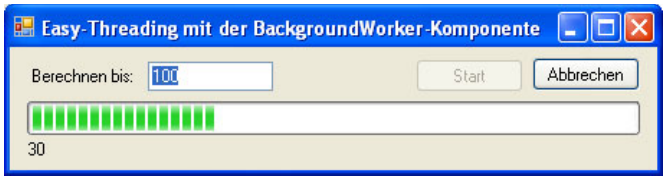


Abbildung 208: Das BackgroundWorker-Beispiel

Zum Kompilieren dieses Beispiels benötigen Sie ein Formular mit einem `BackgroundWorker` mit Namen `backgroundWorker`, einer `TextBox` mit Namen `maxNumberTextBox`, einer `ProgressBar` mit Namen `progressBar`, einem `Label` mit Namen `resultLabel` und zwei Schaltern mit Namen `startButton` und `cancelButton`. Neben der Referenzierung der Assembly *System.Windows.Forms.dll* und der Einbindung des Namensraums `System.Windows.Forms` müssen Sie die Namensräume `System`, `System.Threading` und `System.ComponentModel` importieren.

Zum Starten des `BackgroundWorker` rufen Sie die `RunWorkerAsync`-Methode auf. Dieser Methode können Sie am ersten Argument ein beliebiges Objekt übergeben, das Sie in der Thread-Methode auswerten können. In meinem Beispiel übergebe ich den Text der `TextBox`, deaktiviere zusätzlich den Start- und aktiviere den Abbrechen-Schalter:

```
private void startButton_Click(object sender, EventArgs e)
{
   // Den BackgroundWorker starten
   this.backgroundWorker.RunWorkerAsync(
      this.maxNumberTextBox.Text);

   // Den Start-Schalter deaktivieren und den Abbrechen-Schalter aktivieren
   this.startButton.Enabled = false;
   this.cancelButton.Enabled = true;
}
```

Listing 757: Starten des BackgroundWorkers

Wenn Sie das Abbrechen der Thread-Methode ermöglichen wollen, stellen Sie die `WorkerSupportsCancellation`-Eigenschaft auf `true` ein und rufen in einer entsprechenden Methode einfach die `CancelAsync`-Methode der `BackgroundWorker`-Instanz auf. Damit brechen Sie den Thread nicht direkt ab, sondern setzen die Eigenschaft `CancellationPending`, die Sie im `DoWork`-Ereignis abfragen, auf `true`.

```
private void cancelButton_Click(object sender, EventArgs e)
{
   this.backgroundWorker.CancelAsync();
}
```

Listing 758: Abbrechen des aktuellen Thread

In dem Ereignis DoWork programmieren Sie die Methode, die in dem Thread aufgerufen werden soll. Ein beim Starten des BackgroundWorker-Objekts übergebenes Argument können Sie aus der Eigenschaft Argument des Ereignisargument-Objekts auslesen.

Zur Aktualisierung der Anwendungsoberfläche rufen Sie die ReportProgress-Methode des BackgroundWorker-Objekts auf. Dieser Methode übergeben Sie am ersten Argument einen Prozentwert. Am zweiten Argument (userState) können Sie ein beliebiges Objekt übergeben, dessen Daten Sie im ProgressChanged-Ereignis auswerten können. Voraussetzung dafür, dass der BackgroundWorker eine Fortschrittsmeldung unterstützt, ist, dass die WorkerReportsProgress-Eigenschaft auf true eingestellt ist.

Einen anstehenden Abbruch der Thread-Methode fragen Sie über die CancellationPending-Eigenschaft ab. Ist diese true, setzen Sie die Cancel-Eigenschaft des Ereignisargument-Objekts auf true und brechen die Thread-Methode ab.

Mein einfaches Beispiel liest zunächst das übergebene Argument aus. Ist dieses ungültig, wird eine Exception geworfen. Obwohl Exceptions in einer Thread-Methode normalerweise ignoriert werden, führen diese in der DoWork-Methode eines BackgroundWorker-Objekts dazu, dass das RunWorkerCompleted-Ereignis aufgerufen wird und Sie die Exception aus der Eigenschaft Error des Ereignisargument-Objekts auslesen können.

Zur Signalisierung des Fortschritts übergibt das Beispiel in der Berechnungs-Schleife den aktuellen Fortschritts-Prozentwert und am zweiten Argument lediglich den Wert der Variablen *number*. In der Schleife fragt das Programm die CancellationPending-Eigenschaft der BackgroundWorker-Instanz ab, um zu erkennen, ob der Thread abgebrochen werden soll. Da dieses Beispiel nur der Demonstration dient, wird der Thread in der Schleife für 20 ms angehalten, damit die Berechnung nicht allzu schnell erfolgt (und der Rest des Systems nicht allzu sehr ausgebremst wird).

```
private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{
   // Argument auslesen
   long maxNumber = 0;
   try
   {
      maxNumber = Convert.ToInt64(e.Argument);
   }
   catch (Exception ex)
   {
      throw new ArgumentException("Das Argument der Thread-Methode " +
         "muss eine positive Ganzzahl sein: " + ex.Message);
   }

   long result = 0;
   for (long number = 1; number <= maxNumber; number++)
   {
```

Listing 759: DoWork-Ereignismethode einer BackgroundWorker-Instanz

```
      // Abfragen, ob abgebrochen werden soll
      if (this.backgroundWorker.CancellationPending)
      {
         // Angeben, dass abgebrochen wurde
         e.Cancel = true;

         // und raus aus der Methode
         return;
      }

      // Berechnung ausführen
      result += number;

      // Thread kurz anhalten
      Thread.Sleep(20);

      // Den Fortschritt melden
      int percent = (int)((number / (float)maxNumber) * 100);
      this.backgroundWorker.ReportProgress(percent, number);
   }

   // Das Ergebnis zurückgeben
   e.Result = result;
}
```

Listing 759: DoWork-Ereignismethode einer BackgroundWorker-Instanz (Forts.)

Im Ereignis `ProgressChanged` können Sie die an `ReportProgress` übergebenen Daten threadsicher auswerten. Die an die `ReportProgress`-Methode übergebenen Werte erreichen Sie über die Eigenschaften `ProgressPercentage` und `UserState` des Ereignisargument-Objekts. Mein Beispiel aktualisiert mit dem Prozentwert eine `ProgressBar` und schreibt den am zweiten Argument übergebenen Wert in ein `Label`:

```
private void backgroundWorker_ProgressChanged(object sender,
   ProgressChangedEventArgs e)
{
   // ProgressBar und Label aktualisieren
   this.progressBar.Value = e.ProgressPercentage;
   this.resultLabel.Text = e.UserState.ToString();
}
```

Listing 760: Aktualisieren der Anwendungs-Oberfläche im ProgressChanged-Ereignis einer BackgroundWorker-Instanz

Im Ereignis `RunWorkerCompleted` können Sie auf das Ende der Thread-Methode reagieren. Über die Eigenschaft `Error` des Ereignisargument-Objekts erfahren Sie zunächst, ob während der Ausführung eine Exception aufgetreten ist. Über `Cancelled` können Sie abfragen, ob die Aus-

führung der Methode abgebrochen wurde. Mein Beispiel gibt eine Information über den Zustand aus und schaltet den Status der beiden Schalter wieder zurück:

```
private void backgroundWorker_RunWorkerCompleted(object sender,
   RunWorkerCompletedEventArgs e)
{
   if (e.Error != null)
   {
      // In der Thread-Methode ist ein Fehler aufgetreten
      MessageBox.Show(e.Error.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   else if (e.Cancelled == false)
   {
      // Der Thread wurde abgebrochen
      this.resultLabel.Text =
         "Fertig. Das Ergebnis ist: " + e.Result.ToString();
   }
   else
   {
      // Der Thread wurde normal beendet
      this.resultLabel.Text = "Abgebrochen";
   }

   // Start-Schalter aktivieren und
   // Abbrechen-Schalter deaktivieren
   this.startButton.Enabled = true;
   this.cancelButton.Enabled = false;
}
```

Listing 761: Auswertung des Endes der Thread-Methode

367 Parameter an Threads übergeben und Ergebnisse auslesen

Wenn Sie einen normalen Thread (keinen `BackgroundWorker`-Thread, siehe Rezept 366) programmieren wollen, müssen Sie dazu eine Methode implementieren, die entweder der Signatur des Delegaten `ThreadStart` oder der des Delegaten `ParameterizedThreadStart` entspricht. Die Signatur von `ThreadStart` erlaubt keine Argumente, diejenige von `ParameterizedThreadStart` erlaubt allerdings ein `Object` Argument. An diesem Argument können Sie ein beliebiges Objekt mit Daten übergeben, die Sie innerhalb der Thread-Methode auswerten wollen. Wollen Sie mehrere Daten übergeben, können Sie dazu eine Struktur oder Klasse mit entsprechenden Eigenschaften implementieren und eine Instanz dieser Klasse an die Thread-Methode übergeben. Auf diese Weise ist die Übergabe von Argumenten an eine Thread-Methode sehr einfach.

Das folgende Beispiel demonstriert diese Vorgehensweise an einer Primzahlberechnung. Ein Thread soll alle Primzahlen bis zu einem gegebenen Grenzwert ermitteln und das Ergebnis in die Text-Eigenschaft eines `TextBlock`-Steuerelements schreiben.

Zum Kompilieren dieses (`WPF`-)Beispiels benötigen Sie ein Fenster mit einer `TextBox` mit Namen `maxNumberTextBox`, einem `TextBlock` mit Namen `resultTextBlock`, einem Schalter mit Namen `startButton1` und – für das Beispiel weiter unten – einem Schalter mit dem Namen `startButton2`. Das Beispiel erfordert den Import der Namensräume `System`, `System.Threading`, `System.Windows` und `System.Windows.Controls`.

Zur Übergabe der Argumente habe ich zunächst die Klasse `PrimeNumberCalculatorArguments` entwickelt:

```
private class PrimeNumberCalculatorArguments
{
   /* Der Grenzwert für die Primzahlenberechnung  */
   public long MaxNumber;

   /* Der TextBlock, in den das Ergebnis geschrieben werden soll */
   public TextBlock ResultTextBlock;
}}
```

Listing 762: Klasse für die Argumente des Threads, der über ParameterizedThreadStart gestartet wird

Die Methode für den Thread (`CalculatePrimeNumbers`) besitzt ein `Object`-Argument, an das später das Objekt übergeben wird, das die Argumente zur Ausführung der Methode verwaltet. Dieses Objekt wird innerhalb der Methode in den erwarteten Typ gecastet und ausgewertet. `CalculatePrimeNumbers` geht alle Zahlen zwischen 2 und `MaxValue` durch und überprüft für jede dieser Zahlen (nach dem »Sieb des Eratosthenes«), ob es sich um eine Primzahl handelt. Gefundene Primzahlen werden in dem `TextBlock`-Steuerelement ausgegeben, das über die Eigenschaft `ResultTextBlock` des übergebenen Argument-Objekts referenziert wird. Die Aktualisierung des Steuerelements erfolgt natürlich threadsicher über die `Invoke`-Methode des `Dispatcher`-Objekts des Steuerelements (siehe Rezept 365):

```
private void CalculatePrimeNumbers(object args)
{
   // Casten des übergebenen Objekts
   PrimeNumberCalculatorArguments arguments =
      (PrimeNumberCalculatorArguments)args;

   // Flags-Array, das für jede Zahl verwaltet,
   // ob diese keine Primzahl ist (true = keine Primzahl)
   bool[] numberFlags = new bool[arguments.MaxNumber];
```

Listing 763: Methode für den Thread, der über ParameterizedThreadStart gestartet wird

```
   // Die Zahlen bis zur Wurzel aus der maximalen Zahl durchgehen
   int max = (int)Math.Sqrt(arguments.MaxNumber);
   for (long number = 2; number < max; number++)
   {
      if (numberFlags[number] == false)
      {
         // Die Zahl ist noch nicht als keine Primzahl gekennzeichnet
         for (long divisor = number; divisor < numberFlags.Length; divisor++)
         {
            if (divisor != number && divisor % number == 0)
            {
               // Die Zahl ist keine Primzahl
               numberFlags[divisor] = true;
            }
         }
      }
   }

   // Das Flags-Array durchgehen und die Primzahlen ausgeben
   for (int i = 2; i < numberFlags.Length; ++i)
   {
      if (numberFlags[i] == false)
      {
         // Das Steuerelement threadsicher aktualisieren
         arguments.ResultTextBlock.Dispatcher.Invoke(
            System.Windows.Threading.DispatcherPriority.Normal,
            new Action<TextBlock, string>(
            (textBlock, text) => textBlock.Text += (text + " ")),
            arguments.ResultTextBlock, i.ToString());
      }
   }
}
```

Listing 763: Methode für den Thread, der über ParameterizedThreadStart gestartet wird (Forts.)

In einem Beispiel-WPF-Formular kann der Anwender den Grenzwert eingeben (Abbildung 209) und eine Berechnung starten.

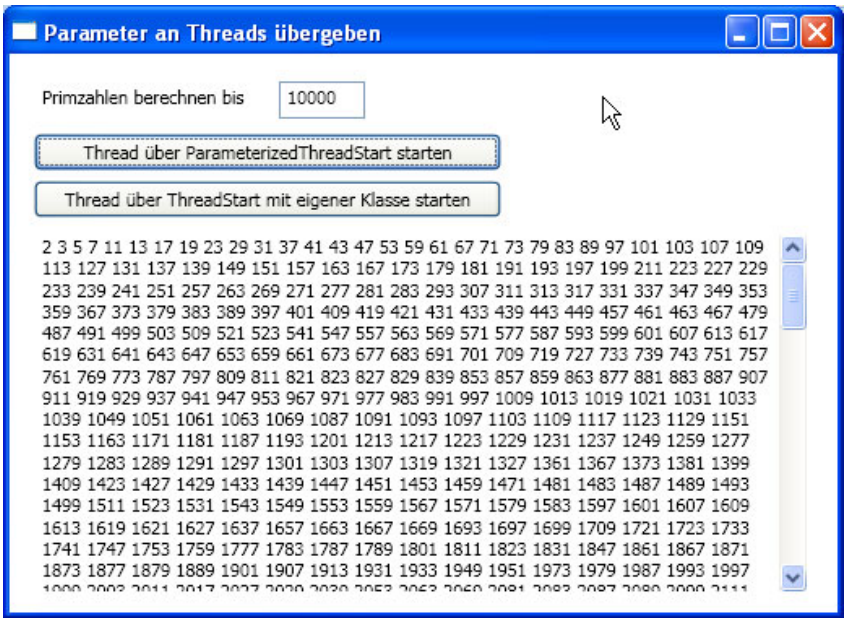


Abbildung 209: Das Formular zur Demonstration der Übergabe von Parametern an einen Thread

In der Ereignisbehandlungsmethode des ersten Schalters wird eine Instanz der `PrimeNumberCalculatorArguments`-Klasse erzeugt und initialisiert. Das in die Eigenschaft `ResultTextBlock` geschriebene Steuerelement ist übrigens ein `TextBlock`, das unter den Schaltern angelegt ist. Danach wird der Thread über die `Start`-Methode einer neuen `Thread`-Instanz gestartet, wobei das Argument-Objekt übergeben wird:

```
private void startButton1_Click(object sender, RoutedEventArgs e)
{
   // Den Ergebnis-TextBlock leeren
   this.resultTextBlock.Text = null;

   // Objekt für die Argumente der Thread-Methode erzeugen und initialisieren
   PrimeNumberCalculatorArguments arguments =
      new PrimeNumberCalculatorArguments {
      MaxNumber = Convert.ToInt64(this.maxNumberTextBox.Text),
      ResultTextBlock = this.resultTextBlock};

   // Thread mit der CalcPrimeNumbers-Methode starten
   Thread thread = new Thread(this.CalculatePrimeNumbers);
```

Listing 764: Starten eines Threads mit der Übergabe von Argumenten über eine Argument-Klasse

```
   thread.IsBackground = true;
   thread.Start(arguments);
}
```

Listing 764: Starten eines Threads mit der Übergabe von Argumenten über eine Argument-Klasse (Forts.)

Hinweis

Das Setzen von `IsBackground` auf `true` bewirkt übrigens, dass der Thread automatisch beendet wird, wenn das Hauptfenster der Anwendung geschlossen wird.

Diese Lösung des Argument-Übergabe-Problems ist in meinen Augen etwas unsauber, da das Argument der Thread-Methode als `Object` deklariert werden muss. Eine andere, vielleicht bessere Lösung dieses Problems ist eine Klasse, die neben den Thread-Argumenten auch die Thread-Methode implementiert. Diese Methode entspricht der `ThreadStart`-Signatur (die keine Argumente erlaubt) und wird später über eine Instanz der Klasse in einem erzeugten Thread gestartet. Da die Thread-Methode Zugriff auf alle Eigenschaften der Klasse hat, können Sie vor dem Start des Threads die Argumente in den dafür vorgesehenen Eigenschaften übergeben und sogar nach dem Ende des Threads aus anderen Eigenschaften Rückgabewerte auslesen.

Das folgende Beispiel demonstriert diese Vorgehensweise an derselben Primzahlberechnung, die ich bereits oben verwendet habe. Dazu habe ich eine Klasse `PrimeNumberCalculator` entwickelt, die die Primzahlberechnung in der Methode `Calculate` ausführt.

```
private class PrimeNumberCalculator
{
   /* Verwaltet den maximalen Wert zur Berechnung der Primzahl */
   public long MaxValue;

   /* Referenz auf das TextBlock-Steuerelement, das das Ergebnis aufnimmt */
   public TextBlock ResultTextBlock;

   /* Methode für den Thread  */
   public void Calculate()
   {
      // Flags-Array, das für jede Zahl verwaltet,
      // ob diese keine Primzahl ist (true = keine Primzahl)
      bool[] numberFlags = new bool[this.MaxValue];

      // Die Zahlen bis zur Wurzel aus der maximalen Zahl durchgehen
      int max = (int)Math.Sqrt(this.MaxValue);
      for (long number = 2; number < max; number++)
      {
```

Listing 765: Klasse für die Ausführung eines Threads unter der Übergabe von Parametern

```
            if (numberFlags[number] == false)
            {
               // Die Zahl ist noch nicht als keine Primzahl gekennzeichnet
               for (long divisor = number; divisor < numberFlags.Length;
                  divisor++)
               {
                  if (divisor != number && divisor % number == 0)
                  {
                     // Die Zahl ist keine Primzahl
                     numberFlags[divisor] = true;
                  }
               }
            }
         }

         // Das Flags-Array durchgehen und die Primzahlen ausgeben
         for (int i = 2; i < numberFlags.Length; ++i)
         {
            if (numberFlags[i] == false)
            {
               // Das Steuerelement threadsicher aktualisieren
               this.ResultTextBlock.Dispatcher.Invoke(
                  System.Windows.Threading.DispatcherPriority.Normal,
                  new Action<TextBlock, string>(
                  (textBlock, text) => textBlock.Text += (text + " ")),
                  this.ResultTextBlock, i.ToString());
            }
         }
      }
   }
```

Listing 765: Klasse für die Ausführung eines Threads unter der Übergabe von Parametern (Forts.)

In der Ereignisbehandlungsmethode des zweiten Schalters des Beispiels in Abbildung 209 wird nun eine Instanz der `PrimeNumberCalculator`-Klasse erzeugt und mit der Grenzwert-Eingabe und einer Referenz auf das Ausgabe-Steuerelement initialisiert. Dann wird der Thread erzeugt, wobei die `Calculate`-Methode des `PrimeNumberCalculator`-Objekts als Threadmethode übergeben wird. Schließlich wird der Thread nur noch gestartet.

```
private void startButton2_Click(object sender, RoutedEventArgs e)
{
   // Den Ergebnis-TextBlock leeren
   this.resultTextBlock.Text = null;
```

Listing 766: Starten eines Threads mit der Übergabe von Argumenten über eine Klasse, die auch die Thread-Methode enthält

```
    // Instanz der PrimeNumberCalculator-Klasse erzeugen und initialisieren
    PrimeNumberCalculator pnc = new PrimeNumberCalculator();
    pnc.MaxValue = Convert.ToInt64(this.maxNumberTextBox.Text);
    pnc.ResultTextBlock = this.resultTextBlock;

    // Thread mit der Calc-Methode des PrimeNumberCalculator-Objekts starten
    Thread thread = new Thread(pnc.Calculate);
    thread.IsBackground = true;
    thread.Start();
}
```

Listing 766: Starten eines Threads mit der Übergabe von Argumenten über eine Klasse, die auch die Thread-Methode enthält (Forts.)

368 Das ereignisbasierte asynchrone Entwurfsmuster implementieren

Bei der Entwicklung von Klassen mit Methoden, deren Ausführung eine signifikant lange Zeit benötigt, ist es häufig sinnvoll, darüber nachzudenken, eine asynchrone Ausführung dieser Methoden zu ermöglichen. Das asynchrone Ausführen von Methoden ist nicht besonders schwierig. Wenn aber die Methoden einen Fortschritt oder ihr erfolgreiches oder fehlerhaftes Beenden melden sollen, ist es mit dem einfachen asynchronen Ausführen nicht mehr getan.

In diesem Fall wäre es gut, wenn die Klasse Methoden zur Verfügung stellt, die direkt asynchron ausgeführt werden können. Prinzipiell können Sie dazu innerhalb der Methoden mit Threads arbeiten, die gegebenenfalls ein Ereignis für den Fortschritt und eines für das Ende aufrufen. Die Programmierung solcher Methoden wäre sehr einfach. Ein Programmierer, der diese Methoden verwendet, muss jedoch auf einen wichtigen Punkt achten, wenn er in den Ereignismethoden Steuerelemente anspricht: Das jeweilige Ereignis wird nicht in dem Thread ausgeführt, der die Steuerelemente erzeugt hat, sondern in dem Thread, der die (Arbeits-) Methode ausführt. Eine goldene Windows-Regel besagt aber, dass nur der Thread, der ein Steuerelement erzeugt hat, auf dieses zugreifen darf. Vergleichen Sie dazu das Rezept 365.

Die Lösung dieses Problems ist die Implementierung des ereignisbasierten asynchronen Entwurfsmusters. Ich beschreibe dieses hier in einer Kurzform, die nicht unbedingt alle Aspekte berücksichtigt. Vergleichen Sie hierzu die Beschreibung dieses Entwurfsmusters in der .NET-Framework-Hilfe (suchen Sie im Index nach »Ereignisbasiertes asynchrones Muster«).

Hinweis

Mir ist bewusst, dass nun einiges an Text folgt, aber das ereignisbasierte asynchrone Entwurfsmuster ist nicht allzu einfach zu verstehen (so war es wenigstens in meinem Fall ☺). Ich denke, dass eine Beschreibung der Arbeitsweise dieses Musters (unter Berücksichtigung des zur Verfügung stehenden Platzes) notwendig ist um dieses verstehen zu können. Ein Beispiel für die Anwendung einer Klasse, die das asynchrone Entwurfsmuster implementiert, finden Sie natürlich am Ende dieses Rezepts, aber auch im Rezept 216.

Das ereignisbasierte asynchrone Entwurfsmuster, das z. B. von der `WebClient`-Klasse aus dem Namensraum `System.Net` implementiert wird (siehe Rezept 215), sieht vor, dass eine Methode in einer synchronen und einer asynchronen Variante existiert. Die synchrone Methode wird ohne Suffix benannt (z. B. *Demo*), die asynchrone mit dem Suffix *Async* (z. B. *DemoAsync*). Beim Aufrufen der asynchronen Methode kehrt diese in der Regel sofort[1] zurück, und das aufrufende Programm kann weiter ausgeführt werden (die Methode wird intern in einem Thread ausgeführt).

Um eine Fortschrittsmeldung zu ermöglichen, kann ein Ereignis implementiert werden, das laut der Vorgabe von Microsoft *<Methodenname>ProgressChanged* genannt wird. Das Ereignisargument-Objekt dieses Ereignisses ist vom Typ `ProgressChangedEventArgs` (oder einer davon abgeleiteten Klasse). Über die Eigenschaft `ProgressPercentage` erhält der Aufrufer eine Information über den Fortschritt der asynchron aufgerufenen Methode.

Für die Benachrichtigung des Beendens der asynchron aufgerufenen Methode sollte ein Ereignis implementiert werden, das *<Methodenname>Completed* genannt wird. Dieses Ereignis wird immer dann aufgerufen, wenn die asynchrone Methode auf irgendeine Art beendet wurde, also bei einem normalen Beenden, beim Auftreten eines Fehlers oder bei einem Abbruch (der im asynchronen Entwurfsmusters ebenfalls möglich ist). Das Ereignisargument-Objekt dieses Ereignisses ist vom Typ `AsyncCompletedEventArgs`. Die Eigenschaft `Error` dieses Typs liefert eine Information darüber, ob während der Ausführung der Methode eine Ausnahme eingetreten ist. Über die Eigenschaft `Cancelled` können Sie erfahren, ob die Methode abgebrochen wurde. Ist `Errors` gleich `null` und `Cancelled` gleich `false`, wurde die Methode erfolgreich beendet.

Das Besondere an diesen Ereignissen ist, dass sie automatisch im richtigen Thread aufgerufen werden. Der Programmierer, der Methoden für diese Ereignisse implementiert, muss also nicht darauf achten, dass diese in dem Thread ausgeführt werden, der die Steuerelemente erzeugt hat, die aktualisiert werden sollen. Leider fordert gerade dieses Feature (das dem verwendenden Programmierer die Arbeit erleichtert) bei der Entwicklung einer Klasse einiges an Arbeit. Bei der Anwendung der Klasse werden wir aber für unsere Mühe belohnt ☺.

Ein weiterer Teil des asynchronen Entwurfsmusters ist, dass es dem Anwender erlaubt sein sollte, die asynchrone Operation abzubrechen. Die Klasse implementiert dazu für jede asynchrone aufzurufende Methode normalerweise eine Methode *<Methodenname>Cancel* (also z. B. *DemoAsyncCancel*). Besitzt eine Klasse nur eine asynchron aufzurufende Methode, oder dient die Abbruch-Methode dem Abbrechen aller asynchronen Operationen, kann diese auch einfach nur *CancelAsync* genannt werden. Bei einem Abbruch wird das *<Methodenname>Completed*-Ereignis aufgerufen, wobei das Ereignisargument `Cancelled` auf `true` steht.

Der letzte hier besprochene Aspekt des asynchronen Entwurfsmusters ist, dass einige asynchrone Methoden einen Mehrfach-Aufruf erlauben und andere einen erneuten Aufruf erst dann, nachdem ein vorheriger beendet wurde. Klassen mit asynchronen Methoden, die keinen Mehrfach-Aufruf erlauben (auch nicht den gleichzeitigen Aufruf verschiedener Methoden), sollten beim Mehrfach-Aufruf ein `InvalidOperationException` werfen und eine `IsBusy`-Eigen-

1. Manchmal auch etwas später, wie zum Beispiel die asynchrone Methoden der `WebClient`-Klasse, die vor dem asynchronen Up- oder Download erst einiges an Vorarbeit leisten.

schaft implementieren, die eine Aussage darüber macht, ob die Klasse gerade mit der Abarbeitung einer asynchronen Methode beschäftigt ist.

Das Prinzip des ereignisbasierten asynchronen Entwurfsmusters ist das Folgende:

- Die synchron und die asynchron auszuführende Methode ruft eine private Arbeitsmethode auf. Die synchrone Methode ruft diese direkt auf, die asynchrone verwendet zum Aufruf die `BeginInvoke`-Methode eines Delegaten, der der Signatur der Arbeitsmethode entspricht. Das ist nichts Besonderes und entspricht dem normalen asynchronen Methodenaufruf. Der Arbeitsmethode wird aber neben den unter normalen Umständen benötigten Argumenten ein `AsyncOperation`-Objekt übergeben. Über die `Post`-Methode dieses Objekts, das mit Hilfe der `CreateOperation`-Methode der `AsyncOperationManager`-Klasse erzeugt werden kann, werden später in der Arbeitsmethode die Ereignisse so aufgerufen, dass diese automatisch im richtigen Thread ausgeführt werden. Die `CreateOperation`-Methode erwartet ein `Object`-Argument. Dieses Argument wird im weiteren Verlauf bis in die Ereignisse durchgereicht und verwaltet beliebige Daten, die der Aufrufer der asynchron auszuführenden Methode übergeben kann (dafür muss diese natürlich ein Argument zur Verfügung stellen). Der Aufrufer kann diese Daten in den Ereignissen weiter verarbeiten.
- Die asynchron aufgerufene Arbeitsmethode ruft zu gegebener Zeit das Fortschritts-Ereignis auf. Dazu verwendet sie die `Post`-Methode des übergebenen `AsyncOperation`-Objekts. Diese Methode sorgt dafür, dass der Aufruf in den richtigen Thread umgeleitet wird. Im Prinzip ist das einfach zu verstehen (natürlich nicht das, was intern geschieht). Die Umsetzung ist aber leider etwas kompliziert. `Post` erwartet nämlich (natürlich) eine Referenz auf einen Delegaten zum Aufruf des Ereignisses. Dieser Delegat muss vom Typ `SendOrPostCallback` sein. Die Referenz kann als privates Feld in der Klasse implementiert werden. Der Delegat muss natürlich eine passende Methode referenzieren, die ebenfalls in der Klasse implementiert ist. Im Konstruktor der Klasse wird die Delegat-Referenz mit einer neuen Instanz des mit dieser Methode initialisierten `SendOrPostCallback`-Delegat initialisiert. Um das Ganze noch ein wenig komplizierter zu machen ruft die Delegat-Methode nicht das Ereignis direkt auf, sondern eine geschützte, virtuelle Methode, die wie das Ereignis benannt ist, allerdings mit dem Präfix *On*. Diese Methode schließlich ruft das Ereignis auf. Dieser Aufruf entspricht dem Ereignisaufruf-Modell, das alle Klassen im .NET Framework implementieren. Damit wird Programmierern, die von Klassen neue Klassen ableiten, ermöglicht, über das Überschreiben dieser Methoden auf Ereignisse zu reagieren. Da den Ereignissen normalerweise Ereignisargumente übergeben werden müssen, werden diese über eine Instanz der entsprechenden Ereignisargument-Klasse von der `Post`-Methode bis an das Ereignis weitergereicht.
- Die `CancelAsync`-Methode setzt lediglich ein boolesches Feld. Dieses Feld wird in der Arbeitsmethode an geeigneter Stelle abgefragt. Ist dieses Feld `true`, bricht die Arbeitsmethode ihre Ausführung ab und ruft das `DemoAsyncCompleted`-Ereignis auf. Das Feld `Cancelled` des übergebenen Ereignisargument-Objekts wird in diesem Fall auf `true` gesetzt.

Abbildung 210 fasst das Ganze schematisch zusammen.

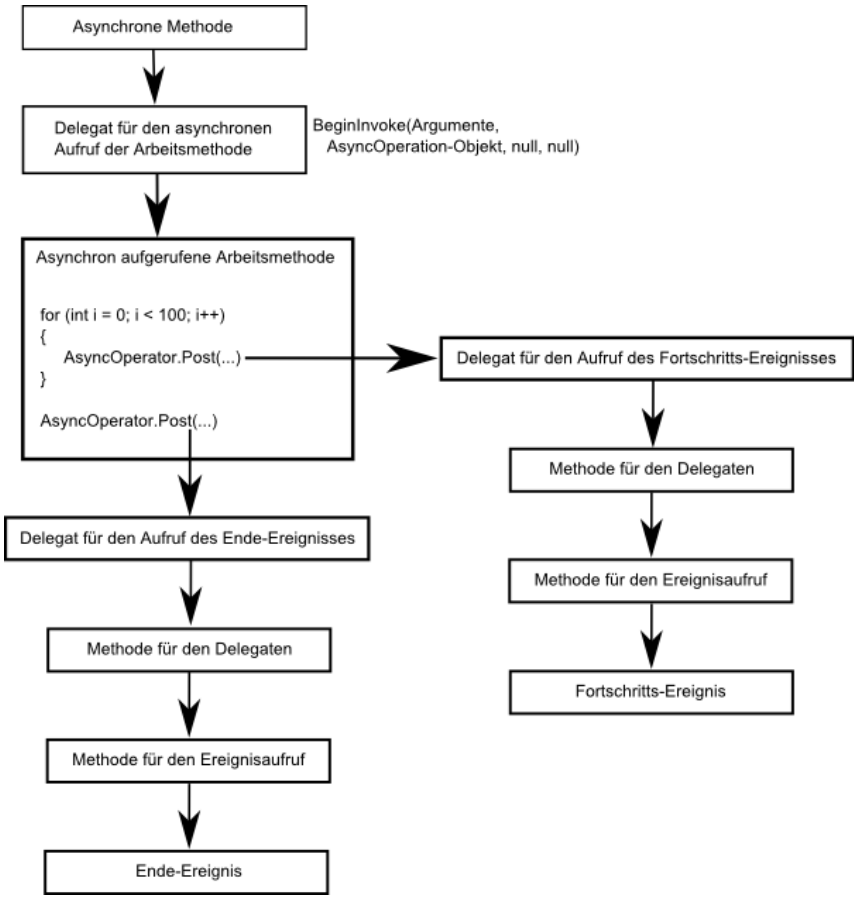


Abbildung 210: Schematische Darstellung des Ablaufs beim Methodenaufruf nach dem ereignisbasierten asynchronen Entwurfsmuster

So, nun folgt Quellcode ☺. Im folgenden Beispiel implementiere ich eine Klasse `AsyncDemo`, die das ereignisbasierte asynchrone Entwurfsmuster in einer praxisorientierten Minimalform implementiert. Die Arbeitsmethode dieser Klasse macht nichts weiter, als eine einfache Schleife mit einer gegebenen Anzahl Durchläufe durchzugehen und in der Schleife die `Sleep`-Methode der `Thread`-Klasse aufzurufen. Dies soll eine länger dauernde Aktion simulieren. Die Methode `Demo` ruft die Arbeitsmethode synchron auf, die Methode `DemoAsync` ruft diese Methode asynchron auf. Beide Methoden erlauben (aus Vereinfachungsgründen) nicht den mehrfachen Aufruf. Deswegen implementiert die Klasse `AsyncDemo` das Busy-Flag `IsBusy`.

Dem Ereignis `DemoAsyncProgressChanged` dieser Klasse soll nicht nur ein Prozentwert für den Fortschritt übergeben werden, sondern auch die Anzahl der insgesamt auszuführenden und die Anzahl der bisher ausgeführten Durchläufe. Deswegen kann der bereits im Namensraum `System.ComponentModel` enthaltene Delegat `ProgressChangedEventHandler` nicht verwendet werden, da dieser nur eine Eigenschaft `ProgressPercentage` in den Ereignisargumenten besitzt. Aus diesem Grunde habe ich eine eigene Ereignisargument-Klasse und einen passenden Delegaten implementiert.

Zum Kompilieren des ab hier beschriebenen Quellcodes müssen Sie die Namensräume `System`, `System.ComponentModel` und `System.Threading` importieren.

```
/* Ereignisargument-Klasse für das DemoAsyncProgressChanged-Ereignis */
public class DemoAsyncProgressChangedEventArgs : ProgressChangedEventArgs
{
   /* Gibt die Anzahl der fertig gestellten Durchläufe an */
   public readonly byte CyclesDone;

   /* Gibt die Anzahl der insgesamt auszuführenden Durchläufe an */
   public readonly byte CyclesTotal;

   /* Konstruktor */
   internal DemoAsyncProgressChangedEventArgs(byte cyclesDone,
      byte cyclesTotal, int progressPercentage, object userState):
      base(progressPercentage, userState)
   {
      this.CyclesTotal = cyclesTotal;
      this.CyclesDone = cyclesDone;
   }
}

/* Delegat für das DemoAsyncProgressChanged-Ereignis */
public delegate void DemoAsyncProgressChangedEventHandler(
   object sender, DemoAsyncProgressChangedEventArgs e);
```

Listing 767: Ereignisargument-Klasse und Delegat für das DemoAsyncProgressChanged-Ereignis

Die Klasse `AsyncDemo`, die das ereignisbasierte asynchrone Entwurfsmuster implementiert, beginnt mit der Deklaration der Eigenschaften und Aufrufmethoden für die Ereignisse, die den Fortschritt und das Beenden der `DemoAsync`-Methode melden. Die Methode `OnDemoAsyncCompleted` setzt neben dem Aufruf des `DemoAsyncCompleted`-Ereignisses auch das Busy-Flag zurück:

```
public class AsyncDemo
{
   /* Wird aufgerufen, wenn der Fortschritt der asynchronen Methode */
   /* sich geändert hat */
   public event DemoAsyncProgressChangedEventHandler
      DemoAsyncProgressChanged;

   /* Wird aufgerufen, wenn die asynchrone Methode beendet wurde */
   public event AsyncCompletedEventHandler DemoAsyncCompleted;
   /* Ruft das DemoAsyncProgressChanged-Ereignis auf */
```

Listing 768: Deklaration der Ereignis-Eigenschaften und Ereignis-Aufrufmethoden für die Ereignisse der AsyncDemo-Klasse

```
protected virtual void OnDemoAsyncProgressChanged(
   DemoAsyncProgressChangedEventArgs e)
{
   // Ereignis aufrufen
   if (this.DemoAsyncProgressChanged != null)
   {
      this.DemoAsyncProgressChanged(this, e);
   }
}

/* Ruft das DemoAsyncCompleted-Ereignis auf
   und setzt das Busy-Flag zurück*/
protected virtual void OnDemoAsyncCompleted(AsyncCompletedEventArgs e)
{
   // Ereignis aufrufen
   if (this.DemoAsyncCompleted != null)
   {
      this.DemoAsyncCompleted(this, e);
   }

   // Busy-Flag zurücksetzen
   this.isBusy = false;
}
```

Listing 768: Deklaration der Ereignis-Eigenschaften und Ereignis-Aufrufmethoden für die Ereignisse der AsyncDemo-Klasse (Forts.)

Dann folgt die Deklaration der `IsBusy`-Eigenschaft und des privaten Feldes `cancellationPending`, das später für den Abbruch der asynchron aufgerufenen Methode verwendet wird:

```
private bool isBusy;
/* Gibt an, ob die asynchrone Methode gerade beschäftigt ist */
public bool IsBusy
{
   get { return this.isBusy; }
}

/* Wird von der CancelAsync gesetzt
  um die asynchrone Ausführung abzubrechen */
private bool cancellationPending;
```

Listing 769: Deklaration der IsBusy-Eigenschaft und eines Feldes zum Abbrechen der asynchron aufgerufenen Methode

Die Arbeitsmethode (`DemoInternal`) wird von `DemoAsync` asynchron aufgerufen. Dazu muss ein Delegat deklariert werden, der die benötigten Argumente besitzt. Der Arbeitsmethode soll die

Anzahl der Durchläufe (`cycleCount`) und eine Referenz auf das verwendete `AsyncOperation`-Objekt übergeben werden. Deswegen muss der Delegat die entsprechende Signatur besitzen:

```
private delegate void DemoInternalCallHandler(
   byte cycleCount, AsyncOperation asyncOperation);
```

Listing 770: Delegat für den asynchronen Aufruf der Arbeitsmethode

Dann folgen Delegat-Felder und Methoden für den Aufruf der Ereignisse. Die Delegat-Felder müssen vom Typ `SendOrPostCallback` sein, damit sie über die `Post`-Methode des `AsyncOperation`-Objekts aufgerufen werden können. Den Methoden wird (dem Delegaten entsprechend) ein Argument vom Typ `Object` übergeben, das die Daten enthält, die an die Ereignisse weitergereicht werden sollen. Beim Aufruf wird aber tatsächlich eine Instanz der Klasse `DemoAsyncProgressChangedEventArgs` bzw. `AsyncCompletedEventArgs` mit den Daten übergeben, die an das Ereignis weitergereicht werden sollen. Innerhalb der Ereignisaufruf-Methoden muss also gecastet werden:

```
/* Über diesen Delegaten ruft die Arbeitsmethode (DemoInternal) das
   DemoAsyncProgressChanged-Ereignis auf */
private SendOrPostCallback demoAsyncProgressChangedHandler;

/* Diese Methode wird über die Delegat-Referenz
   demoAsyncProgressChangedHandler aufgerufen */
private void DemoAsyncProgressChangedCaller(object state)
{
   // Das DemoAsyncCompleted-Ereignis über die Ereignisaufruf-
   // Methode aufrufen
   this.OnDemoAsyncProgressChanged(
      (DemoAsyncProgressChangedEventArgs)state);
}

/* Über diesen Delegaten ruft die Arbeitsmethode (DemoInternal) das
   DemoAsyncCompleted-Ereignis auf */
private SendOrPostCallback demoAsyncCompletedCallerHandler;

/* Diese Methode wird über die Delegat-Referenz
   demoAsyncCompletedCallerHandler aufgerufen */
private void DemoAsyncCompletedCaller(object state)
{
   // Das DemoAsyncCompleted-Ereignis über die Ereignisaufruf-
   // Methode aufrufen
   this.OnDemoAsyncCompleted((AsyncCompletedEventArgs)state);
}
```

Listing 771: Delegat-Felder und Methoden für den Aufruf der Ereignisse

Der Konstruktor der Klasse instanziert die Delegaten:

```
public AsyncDemo()
{
   this.demoAsyncProgressChangedHandler =
      new SendOrPostCallback(this.DemoAsyncProgressChangedCaller);
   this.demoAsyncCompletedCallerHandler =
      new SendOrPostCallback(this.DemoAsyncCompletedCaller);
}
```

Listing 772: Konstruktor der AsyncDemo-Klasse

Dann folgt die private Methode `DemoInternal`, die die Arbeit ausführt. Dieser Methode werden die Anzahl der auszuführenden Durchläufe und das verwendete `AsyncOperation`-Objekt übergeben. Die eigentliche Arbeit wird in dem Beispiel in Listing 773 lediglich simuliert: Eine Schleife geht von Eins bis `cycleCount` durch und ruft `Thread.Sleep(100)` auf um eine länger dauernde Aktion zu simulieren. Der Rest dieser Methode ist für den Aufruf der Ereignisse notwendiger Quellcode, Fehlerbehandlung und die Behandlung eines Abbruchs.

Dazu setzt `DemoInternal` zunächst das Abbruch-Flag zurück. Dann wird zu Demonstrationszwecken ein Fehler simuliert: Wenn `cycleCount` größer ist als 100, wird der Delegaten `demoAsyncCompletedCallerHandler` über die `Post`-Methode des `AsyncOperation`-Objekts aufgerufen. Dabei übergibt `DemoInternal` ein neues `AsyncCompletedEventArgs`-Objekt, dessen `Error`-Eigenschaft (am ersten Argument des Konstruktors) auf eine neu erzeugte `ArgumentException` gesetzt wird. Die Methode wird danach natürlich beendet. Für den Fall, dass kein `AsyncOperation`-Objekt übergeben wurde (was der Fall ist, wenn `DemoInternal` synchron aufgerufen wird), wird die erzeugte `ArgumentException` geworfen. Wichtig dabei ist, dass das `IsBusy`-Flag, das in `AsyncDemo` gesetzt wird, an geeigneter Stelle wieder zurückgesetzt wird.

Danach beginnt die Schleife, die die Arbeit ausführt. Die Schleife ist in einem `try`-Block implementiert, damit Ausnahmen, die während der Ausführung auftreten, abgefangen und (ähnlich wie bei den simulierten Fehlern) über den `demoAsyncCompletedCallerHandler`-Delegaten an das `DemoAsyncCompleted`-Ereignis weitergegeben oder (für den Fall des synchronen Aufrufs) weiter geworfen werden können.

Innerhalb der Schleife reagiert das Programm auf einen Abbruch, indem es das `cancellationPending`-Feld abfragt. Ist dieses `true`, wird über die `Post`-Methode des `AsyncOperation`-Objekts und den Ereignisaufruf-Delegaten das `DemoAsyncCompleted`-Ereignis aufgerufen. Dabei wird dem erzeugten `AsyncCompletedEventArgs`-Objekt am zweiten Argument des Konstruktors `true` übergeben, damit die `Cancelled`-Eigenschaft auf `true` gesetzt wird.

Danach wird der eigentliche Job ausgeführt (was hier ja nur simuliert wird). Am Ende der Schleife meldet `DemoInternal`, natürlich wieder über die `Post`-Methode des `AsyncOperation`-Objekts, über den Delegaten, der über `demoAsyncProgressChangedHandler` referenziert wird, den aktuellen Fortschritt. Schließlich signalisiert die Methode das erfolgreiche Beenden an das `DemoAsyncCompleted`-Ereignis.

```
private void DemoInternal(byte cycleCount, AsyncOperation asyncOperation)
{
   // Das Abbruch-Flag zurücksetzen
   this.cancellationPending = false;

   // Fehler simulieren
   if (cycleCount > 100)
   {
      // Fehler über den Ende-Delegaten melden oder werfen
      ArgumentException ex = new ArgumentException(
         "cycleCount darf nicht größer 0 sein");
      if (asyncOperation != null)
      {
         // Das Ende-Ereignis aufrufen
         asyncOperation.Post(this.demoAsyncCompletedCallerHandler,
             new AsyncCompletedEventArgs(ex, false,
             asyncOperation.UserSuppliedState));

         // und raus
         return;
      }
      else
      {
         // Das Busy-Flag zurücksetzen
         this.isBusy = false;

         // Exception werfen
         throw ex;
      }
   }

   try
   {
      for (byte i = 0; i <= cycleCount; i++)
      {
         if (asyncOperation != null)
         {
            // Auf einen Abbruch reagieren
            if (this.cancellationPending)
            {
               // Das Ende-Ereignis aufrufen
               asyncOperation.Post(this.demoAsyncCompletedCallerHandler,
                  new AsyncCompletedEventArgs(null, true,
                  asyncOperation.UserSuppliedState));
```

Listing 773: Die Arbeitsmethode

```
                // Die Methode beenden
                return;
            }
        }

        // Den eigentlichen Job ausführen.
        // Hier nur als Demo: 100 ms warten
        Thread.Sleep(100);

        if (asyncOperation != null)
        {
            // Den Fortschritt melden
            int progressPercentage = (int)((i / (float)cycleCount) * 100);
            asyncOperation.Post(this.demoAsyncProgressChangedHandler,
                new DemoAsyncProgressChangedEventArgs(i, cycleCount,
                progressPercentage, asyncOperation.UserSuppliedState));
        }
    }

    if (asyncOperation != null)
    {
        // Das Beenden der Methode melden
        asyncOperation.Post(this.demoAsyncCompletedCallerHandler,
            new AsyncCompletedEventArgs(null, false,
            asyncOperation.UserSuppliedState));
    }
}
catch (Exception ex)
{
    // Fehler über den Ende-Delegaten melden oder weiterwerfen
    if (asyncOperation != null)
    {
        // Das Ende-Ereignis aufrufen
        asyncOperation.Post(this.demoAsyncCompletedCallerHandler,
            new AsyncCompletedEventArgs(ex, false,
            asyncOperation.UserSuppliedState));
    }
    else
    {
        // Das Busy-Flag zurücksetzen
        this.isBusy = false;

        // Exception weiterwerfen
```

Listing 773: Die Arbeitsmethode (Forts.)

```
            throw ex;
         }
      }
   }
```

Listing 773: Die Arbeitsmethode (Forts.)

Nun folgen noch die Methoden zum synchronen und asynchronen Ausführen der Arbeitsmethode und die Methode zum Abbrechen. Die Methode `Demo` führt den Job synchron aus und ruft dazu einfach `DemoInternal` auf, indem am zweiten Argument (der `AsyncOperation`-Referenz) `null` übergeben wird. Dabei wird aber das Busy-Flag abgefragt, damit diese Methode nicht aufgerufen werden kann, während die Arbeitsmethode gerade noch asynchron ausgeführt wird. Demo wirft dazu eine `InvalidOperationException`, falls die `IsBusy`-Eigenschaft `true` ist. Außerdem wird dieses Flag vor dem Aufruf der Arbeitsmethode gesetzt und nach dem Aufruf wieder zurückgesetzt, um zu verhindern, dass die asynchrone Variante aufgerufen wird, während die synchrone (in einem Thread) ausgeführt wird. Sicher ist sicher ...

Die erste Variante von `DemoAsync` (ohne Argument für die Benutzerdaten) ruft lediglich die zweite Variante auf, der neben der Anzahl der auszuführenden Durchläufe auch das Objekt mit den Benutzerdaten übergeben wird. Diese Variante fragt ähnlich der `Demo`-Methode zunächst die `IsBusy`-Eigenschaft ab um den wiederholten asynchronen Aufruf zu verhindern. Dieses Flag wird dann auf `true` gesetzt.

Nun folgt der wichtige Teil, nämlich das asynchrone Ausführen der Arbeitsmethode. Dazu erzeugt `DemoAsync` ein neues `AsyncOperation`-Objekt, dem die Benutzerdaten übergeben werden. Über die `BeginInvoke`-Methode einer neuen Instanz des `WorkerEventHandler`-Delegaten ruft `DemoAsync` dann die `DemoInternal`-Methode asynchron auf. Dabei werden am ersten Argument die Anzahl der auszuführenden Durchläufe, am zweiten Argument das `AsyncOperation`-Objekt und an den in unserem Fall nicht benötigten folgenden Argumenten (`callback` und `@object`[2]) `null` übergeben.

Die Methode `CancelAsync` schließlich setzt lediglich das Feld `cancellationPending`:

```
/* Beispiel-Methode, die die auszuführende Arbeit synchron ausführt */
public void Demo(byte cycleCount)
{
   // Überprüfen, ob gerade eine asynchrone Operation ausgeführt wird
   if (this.isBusy)
   {
      throw new InvalidOperationException(
         "Demo kann nicht aufgerufen werden, wenn DemoAsync " +
         "gerade ausgeführt wird");
   }
```

Listing 774: Methoden zum Aufruf und zum Abbruch der Arbeitsmethode

2. Das @ vor *object* hat die Bedeutung, dass damit Bezeichner deklariert werden können, die denselben Namen besitzen wie Schlüsselwörter.

```
      try
      {
         // Das Busy-Flag setzen
         this.isBusy = true;

         // Die private Methode synchron aufrufen
         this.DemoInternal(cycleCount, null);
      }
      finally
      {
         // Das Busy-Flag zurücksetzen
         this.isBusy = false;
      }
   }

   /* Beispiel-Methode, die die auszuführende Arbeit asynchron ausführt */
   public void DemoAsync(byte cycleCount)
   {
      this.DemoAsync(cycleCount, null);
   }

   /* Beispiel-Methode, die die auszuführende Arbeit asynchron ausführt */
   public void DemoAsync(byte cycleCount, object userState)
   {
      // Überprüfen, ob gerade eine asynchrone Operation ausgeführt wird
      if (this.isBusy)
      {
         throw new InvalidOperationException(
            "DemoAsync kann nicht mehrfach gleichzeitig aufgerufen werden");
      }

      // Busy-Flag setzen
      this.isBusy = true;

      // AsyncOperation-Objekt erzeugen, über das die Ereignis-Handler
      // im richtigen Thread aufgerufen werden können
      AsyncOperation asyncOperation =
         AsyncOperationManager.CreateOperation(userState);

      // Die private Methode asynchron aufrufen
      WorkerEventHandler workerEventHandler =
         new WorkerEventHandler(this.DemoInternal);
      workerEventHandler.BeginInvoke(cycleCount, asyncOperation, null, null);
   }
```

Listing 774: Methoden zum Aufruf und zum Abbruch der Arbeitsmethode (Forts.)

```
   /* Bricht eine aktuelle asynchrone Operation ab */
   public void CancelAsync()
   {
      this.cancellationPending = true;
   }
}
```

Listing 774: Methoden zum Aufruf und zum Abbruch der Arbeitsmethode (Forts.)

Damit ist die Beispiel-Klasse für das ereignisbasierte asynchrone Entwurfsmuster beendet.

Tipp

Wenn Sie die Klasse (aus dem Repository) für Ihre Zwecke einsetzen wollen, kopieren Sie den gesamten Quellcode, ersetzen Sie »Demo« durch Ihren eigenen Namen, ändern Sie die Ereignisargument-Klasse entsprechend Ihren Anforderungen ab, und programmieren Sie die private Methode (`DemoInternal`) um.

Nun folgt lediglich noch ein Beispiel für die Anwendung einer Instanz dieser Klasse. Und diese ist (im Gegensatz zur Implementierung der Klasse) einfach: Sie erzeugen dazu eine Instanz der Klasse und rufen deren Methoden auf. Wenn Sie die asynchrone Variante verwenden wollen, weisen Sie den Ereignis-Eigenschaften zuvor Instanzen der entsprechenden Delegaten zu. Visual Studio 2008 hilft ja dabei. In den Methoden zu diesen Ereignissen können Sie beliebig programmieren.

Das folgende Beispiel ist in einer einfachen `Windows.Forms`-Anwendung implementiert (Abbildung 211). In einer `TextBox` (`repeatCountTextBox`) kann der Anwender die Anzahl Durchläufe eingeben. Über den ersten Schalter (`callDemoSyncButton`) wird die synchrone Methode `Demo` aufgerufen. Der zweite Schalter (`callDemoAsyncButton`) ruft die Methode `DemoAsync` auf. Über den dritten Schalter (`cancelButton`) kann der Anwender die Ausführung der asynchronen Methode abbrechen. Über eine `ProgressBar` (`progressBar`) wird der prozentuale Fortschritt gemeldet, ein `ToolStripStatusLabel` (`toolStripStatusLabel`) in einer `StatusStrip`-Instanz dient einer Status-Meldung.

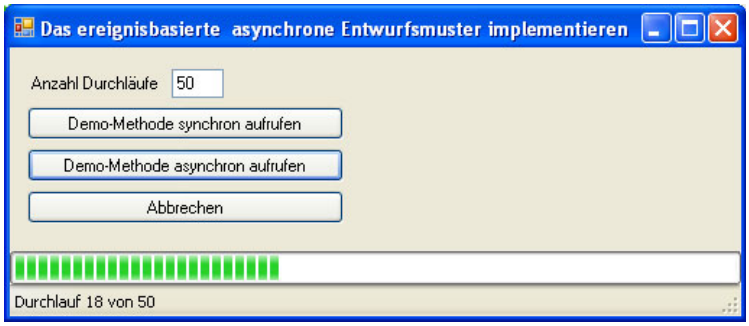


Abbildung 211: Beispiel-Programm zur Anwendung der Demo-Klasse für das ereignisbasierte asynchrone Entwurfsmuster

Der Quellcode sieht folgendermaßen aus:

```
/* Referenz auf die Instanz der Demo-Klasse */
AsyncDemo asyncDemo;

/* Konstruktor */
public StartForm()
{
   InitializeComponent();

   // AsyncDemo-Instanz erzeugen
   this.asyncDemo = new AsyncDemo();

   // Ereignisse zuweisen
   this.asyncDemo.DemoAsyncProgressChanged +=
      new DemoAsyncProgressChangedEventHandler(
      this.demo_AsyncProgressChanged);
   this.asyncDemo.DemoAsyncCompleted +=
      new AsyncCompletedEventHandler(this.demo_AsyncCompleted);
}

/* Aufruf der synchronen Methode */
private void callDemoSyncButton_Click(object sender, EventArgs e)
{
   try
   {
      this.toolStripStatusLabel.Text = "Arbeite ...";
      this.Refresh();

      // Die synchrone Methode aufrufen
      this.asyncDemo.Demo(Convert.ToByte(this.repeatCountTextBox.Text));
      // Den Erfolg melden
      this.toolStripStatusLabel.Text = "Fertig";
      MessageBox.Show("Fertig", Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Information);
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}

/* Aufruf der asynchronen Methode */
private void callDemoAsyncButton_Click(object sender, EventArgs e)
```

Listing 775: Quellcode der Methoden im Formular zur Anwendung der Demo-Klasse für das ereignisbasierte asynchrone Entwurfsmuster

```
{
   try
   {
      this.toolStripStatusLabel.Text = "Arbeite ...";
      this.Refresh();
      this.cancelButton.Enabled = true;

      // Die asynchrone Methode aufrufen
      this.asyncDemo.DemoAsync(
         Convert.ToByte(this.repeatCountTextBox.Text), null);
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}

/* Bricht die aktuelle asynchrone Operation ab */
private void cancelButton_Click(object sender, EventArgs e)
{
   this.asyncDemo.CancelAsync();
}

/* Wird aufgerufen, wenn sich der Fortschritt bei der Abarbeitung der
   asynchronen Methode geändert hat */
private void demo_AsyncProgressChanged(object sender, DemoAsyncProgressChangedEventArgs
e)
{
   // Den Fortschritt melden
   this.toolStripStatusLabel.Text = "Durchlauf " +
      e.CyclesDone + " von " + e.CyclesTotal;
   this.progressBar.Value = e.ProgressPercentage;
}

/* Wird aufgerufen, wenn die asynchron aufgerufene Methode beendet wurde */
private void demo_AsyncCompleted(object sender, AsyncCompletedEventArgs e)
{
   if (e.Error != null)
   {
      // Fehler melden
      this.toolStripStatusLabel.Text = "Fehler";
      MessageBox.Show(e.Error.Message, Application.ProductName,
         MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
```

Listing 775: Quellcode der Methoden im Formular zur Anwendung der Demo-Klasse für das ereignisbasierte asynchrone Entwurfsmuster (Forts.)

```
   else if (e.Cancelled)
   {
      // Melden, was abgebrochen wurde
      this.toolStripStatusLabel.Text = "Abgebrochen";
   }
   else
   {
      // Den Erfolg melden
      this.toolStripStatusLabel.Text = "Fertig";
      MessageBox.Show("Fertig", Application.ProductName,
        MessageBoxButtons.OK, MessageBoxIcon.Information);
   }
   this.progressBar.Value = 0;
   this.cancelButton.Enabled = false;
}
```

Listing 775: Quellcode der Methoden im Formular zur Anwendung der Demo-Klasse für das ereignisbasierte asynchrone Entwurfsmuster (Forts.)

Zum Kompilieren dieses Beispiels benötigen Sie ein Formular mit den genannten Steuerelementen und den üblichen Importen. Viel Spaß beim Ausprobieren ☺.

Datenbank-Programmierung

369 Die Anzahl der Datensätze ermitteln, die eine Abfrage in einer Datenbank ergibt

Wenn Sie mit ADO.NET direkt auf Datenbanken zugreifen und Daten über ein `DataReader`-Objekt abfragen, können Sie nicht ermitteln, wie viele Datensätze von der Abfrage zurückgegeben wurden, da ein `DataReader`-Objekt keine entsprechende Eigenschaft besitzt. Die Eigenschaft `RecordsAffected`, die eigentlich die Anzahl der betroffenen Datensätze (allerdings laut Dokumentation nur für Einfüge-, Aktualisierungs- und Löschaktionen) verwaltet, gibt beim Lesen von Daten immer -1 zurück.

Sie können aber recht einfach über die SQL-Funktion `COUNT` ermitteln, wie viele Datensätze von einer Abfrage betroffen sind. Idealerweise fragen Sie die Anzahl vor der eigentlichen Abfrage ab, da viele Datenbanksysteme `COUNT` nur dann in einer Abfrage mit Tabellenfeldern erlauben, wenn diese Felder in einer `GROUP BY`-Klausel angegeben werden. `COUNT` würde sich dann nicht auf die Gesamtzahl der Datensätze beziehen, sondern auf die Datensätze einer Gruppe. Das Ergebnis wäre in diesem Fall falsch.

Listing 776 zeigt das Abfragen der Anzahl der Datensätze am Beispiel der *Products*-Tabelle der *Northwind*-Datenbank im SQL Server. Da die Abfrage mit `COUNT(*)` immer nur und genau einen Datensatz ergibt, fragt das Beispiel die Anzahl über die `ExecuteScalar`-Methode eines `SqlCommand`-Objekts ab. Diese Methode ergibt den Wert der ersten Spalte des ersten Datensatzes als `object`-Typ.

Zum Kompilieren dieses Programms müssen Sie die Namensräume `System` und `System.Data.SqlClient` importieren. Den Verbindungsstring, der bei der Erzeugung des `SqlConnection`-Objekts übergeben wird, müssen Sie natürlich an Ihr System anpassen.

```
// Verbindung zur Northwind-Datenbank aufbauen, die im
// Anwendungsordner gespeichert ist
string connectionString = @"Data Source=.\SQLEXPRESS;" +
   @"AttachDbFilename=|DataDirectory|\Northwind.MDF;" +
   @"Integrated Security=True;User Instance=True";
   using (SqlConnection connection = new SqlConnection(connectionString))
{
   connection.Open();

   // Produkte abfragen
   SqlCommand command = new SqlCommand("SELECT COUNT(*) AS Count " +
      " FROM Products WHERE CategoryId = 1", connection);

   Console.WriteLine("Die Kategorie 1 beinhaltet {0} Artikel",
      command.ExecuteScalar());
}
```

Listing 776: Abfragen der Anzahl der Datensätze, die eine Abfrage ergibt

370 Datenbanken erzeugen

Datenbanken können Sie im Wesentlichen auf zwei Arten erzeugen. Wenn Sie mit LINQ to SQL arbeiten, können Sie ganz einfach die `CreateDatabase`-Methode des `DataContext`-Objekts verwenden. Wenn Sie mit ADO.NET direkt arbeiten, können Sie Datenbanken mit SQL erzeugen.

LINQ to SQL

LINQ to SQL macht das Erzeugen von Datenbanken auf dem System, auf dem Ihre Anwendungen später ausgeführt werden, sehr einfach. Dazu erzeugen Sie zunächst auf Ihrem System ein LINQ-to-SQL-Modell (über den LINQ-to-SQL-Designer in Visual Studio). Sie können das Modell komplett von Hand im Designer erzeugen. Einfacher ist aber ggf. die Erzeugung der Datenbank in einem dafür vorgesehenen Werkzeug (wie dem SQL Server Management Studio) und das Erzeugen des LINQ-to-SQL-Modells über das Ziehen der Tabellen vom Visual-Studio-Server-Explorer in den LINQ-to-SQL-Designer.

Im Programm müssen Sie dann lediglich eine Instanz des `DataContext`-Objekts erzeugen, über die `DatabaseExists`-Methode abfragen, ob die Datenbank (die über den Verbindungsstring des `DataContext`-Objekts definiert ist) existiert und im negativen Fall die `CreateDatabase`-Methode aufrufen um die Datenbank zu erzeugen. Das folgende Beispiel zeigt dies an einem `DataContext`-Objekt, das für die Datenbank erzeugt wurde, die auch im ADO.NET-Abschnitt erzeugt wird:

```
// Den DataContext erzeugen
BookstoreDataContext dataContext = new BookstoreDataContext();

// Die Datenbank erzeugen falls diese nicht existiert
if (dataContext.DatabaseExists() == false)
{
   dataContext.CreateDatabase();
}
```

Listing 777: Erzeugen einer Datenbank über LINQ to SQL

ADO.NET

ADO.NET bietet einen hervorragenden Support zur Abfrage und zur Bearbeitung von Datenbanken. Was jedoch fehlt, ist die Möglichkeit, Datenbanken und die darin enthaltenen Tabellen zu erzeugen. Dazu können Sie aber einfach SQL verwenden. Das folgende Beispiel erzeugt über SQL eine Datenbank *Bookstore* im lokalen SQL Server. Dabei werden drei Tabellen *Authors*, *Books* und *BookAuthors* erzeugt und miteinander in Beziehung gesetzt. Das Schema dieser Datenbank wird in Abbildung 212 dargestellt.

Tipp

An der Adresse *www.juergen-bayer.net/menu.aspx?mainMenu=articles&subMenu=database* finden Sie Artikel über SQL und den SQL Server. In diesen Artikeln können Sie alles Wichtige zum Erzeugen und Verwalten von Datenbanken nachlesen.

Zum Kompilieren des Beispielprogramms müssen Sie die Namensräume `System`, `System.Data` und `System.Data.SqlClient` importieren. Im Verbindungsstring müssen Sie natürlich Ihren Server angeben.

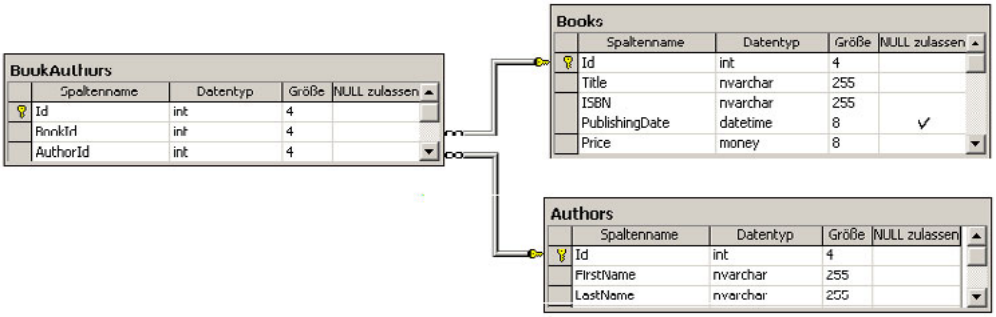


Abbildung 212: Ein SQL-Server-Diagramm der erzeugten Datenbank

Zuerst wird eine Verbindung zum SQL Server aufgebaut. Dabei geben Sie keine Datenbank an:

```
connection = new SqlConnection(
   @"Server=(local)\SQLEXPRESS;Trusted_Connection=Yes");
connection.Open();
```

Listing 778: Aufbau einer Verbindung zum lokalen SQL Server Express

Dann wird zunächst die Datenbank angelegt. Dazu verwenden Sie die SQL-Anweisung `CREATE DATABASE`:

```
string databaseName = "Bookstore";
string sql = "CREATE Database " + databaseName;
SqlCommand command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
```

Listing 779: Anlegen der Datenbank

Bei vielen Datenbanksystemen wie dem SQL Server erlaubt `CREATE DATABASE` noch spezielle Einstellungen, wie z. B. die Festlegung, in welchen Dateien die Datenbank gespeichert werden soll. Lesen Sie diese in der Dokumentation (oder für den SQL Server im oben genannten Artikel) nach. Einige wenige Datenbanksysteme wie z. B. Oracle kennen die `CREATE DATABASE`-Anweisung nicht und verwenden stattdessen `CREATE SCHEMA`.

Dann müssen Sie die Verbindung schließen und zu der neuen Datenbank wieder aufbauen:

```
connection.Close();
connection = new SqlConnection(
   @"Server=(local)\SQLEXPRESS;Database=" + databaseName + ";" +
   "Trusted_Connection=Yes");
connection.Open();
```

Listing 780: Aufbau einer Verbindung zur neuen Datenbank

Nun können Sie die einzelnen Tabellen erzeugen. Dazu verwenden Sie die CREATE TABLE-Anweisung. Ich schreibe die SQL-Befehle zum Erzeugen der Tabellen zunächst in einzelne String-Variablen, um die Befehle später alle gemeinsam ausführen zu können:

```
string sql1 = "CREATE TABLE Authors (" +
   "Id int NOT NULL PRIMARY KEY IDENTITY," +
   "FirstName nvarchar(255) NOT NULL," +
   "LastName nvarchar(255) NOT NULL)";

string sql2 = "CREATE TABLE Books (" +
   "Id int NOT NULL PRIMARY KEY IDENTITY," +
   "Title nvarchar(255) NOT NULL," +
   "ISBN nvarchar(255) NOT NULL," +
   "PublishingDate datetime, " +
   "Price money NOT NULL DEFAULT 0)";

string sql3 = "CREATE TABLE BookAuthors (" +
   "Id int NOT NULL PRIMARY KEY," +
   "BookId int NOT NULL," +
   "AuthorId int NOT NULL)";
```

Listing 781: Erzeugen von Tabellen

In den Klammern geben Sie jeweils die zu erzeugenden Spalten an. Eine Spaltenbeschreibung beginnt mit dem Namen der Spalte gefolgt vom Datentyp. Die Datentypbezeichner sind natürlich datenbankabhängig. Der SQL Server kennt eine Vielzahl an Datentypen, die Sie in der Dokumentation oder im oben genannten Artikel nachlesen können. Die wichtigsten sind:

- `int`: 32-Bit-Integerwert,
- `float`[*n*]: Fließkommazahl mit 1 bis 53 Dezimalstellen, die in *n* spezifiziert werden können. Geben Sie n nicht an, werden 24 Dezimalstellen verwendet,
- `nvarchar`(*n*): Unicode-Zeichen mit einer Maximalanzahl, die in *n* angegeben ist. Sie können bis zu 4.000 Zeichen in einem solchen Feld verwalten. Da nur die tatsächlich benutzten Zeichen gespeichert werden, spielt die Angabe der Maximalgröße für den Speicherverbrauch keine Rolle,
- `datetime`: Datums-/Zeitwert mit der Basis 1.1.1900,
- `money`: Währungsdaten.

Über den Zusatz `NOT NULL` legen Sie fest, dass das Feld keine Nullwerte erlaubt. Der Zusatz `DEFAULT` kennzeichnet Defaultwerte, die bei neuen Datensätzen automatisch eingetragen werden, wenn das Feld nicht gefüllt wird. `PRIMARY KEY` legt die Spalte als Primärschlüssel fest. Über `IDENTITY` können Sie bestimmen, dass der Wert dieser Spalte automatisch beim Einfügen von neuen Datensätzen um 1 erhöht in die Spalte geschrieben wird (damit erzeugen Sie so genannte »Autowert«-Spalten).

Nach der Definition der SQL-Strings zum Erzeugen der Tabellen legt das Beispiel die Beziehungen zwischen diesen fest. Dazu verwenden Sie den SQL-Befehl `ALTER TABLE` mit dem Unterbefehl `ADD CONSTRAINT`:

```
string sql4 = "ALTER TABLE BookAuthors " +
   "ADD CONSTRAINT FK_Books FOREIGN KEY " +
   "(BookId) REFERENCES Books(Id)";
string sql5 = "ALTER TABLE BookAuthors " +
   "ADD CONSTRAINT FK_Authors FOREIGN KEY " +
   "(AuthorId) REFERENCES Authors(Id)";
```

Listing 782: Festlegen der Beziehungen zwischen den Tabellen

Der Zusatz `FOREIGN KEY` sagt aus, dass es sich um einen Fremdschlüssel handelt (der eben für Beziehungen gedacht ist). Hinter `FOREIGN KEY` wird die Spalte angegeben, auf die sich der Fremdschlüssel bezieht, dann folgt `REFERENCES` mit der Angabe der referenzierten Tabelle und der referenzierten Spalte.

Schließlich müssen Sie die einzelnen Strings nur noch zusammensetzen, wobei Sie beim SQL Server einen Zeilenumbruch als Trennung einbauen müssen. Die meisten anderen Datenbanksysteme erfordern ein Semikolon als Trennzeichen. Dann können Sie den SQL-Befehl ausführen (und hoffen, dass dessen Syntax korrekt ist ☺):

```
sql = sql1 + "\r\n" + sql2 + "\r\n" + sql3 + "\r\n" +
   sql4 + "\r\n" + sql5;
command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();

connection.Close();
```

Listing 783: Ausführen der SQL-Befehle zur Erzeugung der Datenbank

371 Abfragen der automatisch vergebenen Id eines neuen Datensatzes

Viele Datenbanksysteme erlauben es, eine Spalte in einer Tabelle als Identity- oder Autowert-Spalte zu deklarieren. Meist wird dazu die Primärschlüssel-Spalte verwendet. Beim SQL Server setzen Sie dazu den Zusatz `IDENTITY` für die betreffende Spalte bei der Erzeugung der Tabelle ein. Der numerische Wert dieser Spalte wird dann beim Einfügen von Datensätzen automatisch um einen bestimmten Inkrementwert (meist 1) erhöht.

Solche Spalten sind in der Praxis immer problematisch: Wenn Datensätze in verknüpften Tabellen eingefügt werden, muss zunächst der Datensatz in der Mastertabelle erzeugt werden. Danach können Sie die Datensätze in der Detailtabelle erzeugen. Das Problem ist, dass Sie dann den Primärschlüsselwert des neuen Master-Datensatzes benötigen.

LINQ to SQL

Wenn Sie Datenbanken mit LINQ to SQL bearbeiten, haben Sie keine Probleme mit dem Abfragen einer automatisch vergebenen ID, denn LINQ to SQL fragt diese nach dem Hinzufügen von Datensätzen automatisch ab und Sie können die neue ID nach dem Hinzufügen des Datensatzes aus der entsprechenden Eigenschaft der Daten-Klasse auslesen:

```
// DataContext erzeugen
BookstoreDataContext dataContext = new BookstoreDataContext();
// Einen Autoren hinzufügen
Author newAuthor = new Author { FirstName = "Matt", LastName = "Ruff" };
dataContext.Authors.InsertOnSubmit(newAuthor);
dataContext.SubmitChanges();

// Den Id-Wert auslesen
int identityValue = newAuthor.Id;
```

Listing 784: Abfragen der automatisch vergebenen ID eines neu hinzugefügten Datensatzes in LINQ to SQL

Hinweis

Falls Sie das Rezept nachvollziehen und Probleme mit den Namen haben, die der LINQ-to-SQL-Designer in der deutschen Visual-Studio-Version für Datenbanken mit englischen Namen erzeugt, lesen Sie das Rezept 288.

ADO.NET

Wenn Sie mit ADO.NET arbeiten, müssen Sie etwas tricksen um automatisch vergebene ID-Werte auslesen zu können. Beim SQL Server können Sie den Wert eines `IDENTITY`-Feldes nach dem Hinzufügen eines Datensatzes über eine Abfrage der globalen Variablen `@@IDENTITY` ermitteln. Diese in Newsgroups häufig beschriebene Lösung ist allerdings problematisch: `@@IDENTITY` ergibt immer den Wert der `IDENTITY`-Spalte des zuletzt eingefügten Datensatzes. Wenn das Einfügen eines Datensatzes einen Trigger auslöst, der wiederum Datensätze in Tabellen mit `IDENTITY`-Spalte einfügt, ergibt `@@IDENTITY` den Wert der `IDENTITY`-Spalte des vom Trigger eingefügten Datensatzes. Außerdem könnte `@@IDENTITY` auch problematisch werden, wenn zwischen dem Einfügen und dem Abfragen Datensätze durch andere Verbindungen zur Datenbank eingefügt werden (was ich allerdings nicht nachvollziehen konnte).

Die Lösung dieses Problems ist die Verwendung der Funktion `SCOPE_IDENTITY`, die den Id-Wert bezogen auf den Gültigkeitsbereich zurückgibt, aus dem heraus das Einfügen erfolgte. Als Gültigkeitsbereich gelten z. B. Trigger, Stored Procedures oder Verbindungen. Fügt also eine Verbindung zur Datenbank einen Datensatz ein, führt `SCOPE_IDENTITY` zum korrekten Wert, auch wenn zwischen dem Einfügen und der Abfrage des ID-Werts eine andere Verbindung oder ein Trigger einen weiteren Datensatz eingefügt hat.

Als Beispiel verwende ich die Datenbank, die im Rezept 370 erzeugt wurde. Das Beispiel setzt SQL zum Einfügen des Datensatzes ein. Sie können `SCOPE_IDENTITY` aber auch nach dem Hinzufügen eines Datensatzes über ein `DataSet` oder `DataTable` abfragen, wobei Sie natürlich da-

rauf achten müssen, dass Sie im `DataSet`- bzw. `DataTable`-Objekt nicht gleich mehrere Master-Datensätze ändern und in die Datenbank schreiben.

Zum Kompilieren dieses Beispiels müssen Sie die Namensräume `System`, `System.Data` und `System.Data.SqlClient` importieren. Den Verbindungsstring, der bei der Erzeugung des `SqlConnection`-Objekts übergeben wird, müssen Sie natürlich an Ihren Server anpassen.

```
// Verbindung zur Bookstore-Datenbank auf dem lokalen
// SQL Server Express aufbauen
   using (SqlConnection connection = new SqlConnection(@"Server=(local)\SQLEXPRESS;" +
      Database=Bookstore;Trusted_Connection=Yes"))
{
   connection.Open();

   // Autor hinzufügen
   string sql = "INSERT INTO Authors (FirstName, LastName) " +
      "VALUES ('Matt', 'Ruff')";
   SqlCommand command = new SqlCommand(sql, connection);
   command.ExecuteNonQuery();

   // Den Id-Wert auslesen
   sql = "SELECT SCOPE_IDENTITY()";
   command = new SqlCommand(sql, connection);
   int identityValue = Convert.ToInt32(command.ExecuteScalar());

   Console.WriteLine("Id des neuen Autors: {0}", identityValue);
}
```

Listing 785: Abfragen der SCOPE_IDENTITY-Funktion zur Ermittlung der Id eines neuen Datensatzes

Andere Datenbanksysteme kennen u. U. auch eine ähnliche Funktion oder Variable, die Sie genauso einfach abfragen können. Lesen Sie in der Dokumentation des Datenbanksystems nach ☺.

372 Bilder und andere binäre Daten in einer Datenbank verwalten

Die meisten Datenbanksysteme kennen einen Feldtyp, der es erlaubt, binäre Daten zu speichern. Beim SQL Server heißt dieser Typ (etwas irreführend) `image`. Im Allgemeinen werden solche Felder allerdings als BLOB-Felder (Binary Large Object) bezeichnet.

In einem BLOB-Feld können Sie beliebige binäre Daten ablegen. Dieses Rezept zeigt, wie Sie das für LINQ to SQL und für ADO.NET programmieren. Die Beispiele basieren auf einer Datenbank mit Namen *Persons* mit einer Tabelle, die mit SQL folgendermaßen erzeugt werden kann:

```
CREATE TABLE Persons (
   Id int PRIMARY KEY NOT NULL IDENTITY,
   FirstName nvarchar(255),
   LastName nvarchar(255),
   Picture image)
```

Das Feld *Id* wird beim Hinzufügen von Datensätzen automatisch um 1 erhöht. Das Feld *Picture* ist vom Typ `image` und verwaltet deswegen binäre Daten.

LINQ to SQL

LINQ to SQL repräsentiert Felder mit binären Daten über den Typ `Binary`. Ein `Binary`-Objekt verwaltet binäre Daten als `Byte`-Array. Beim Schreiben erzeugen Sie eine neue Instanz der `Binary`-Klasse und übergeben das `Byte`-Array am Konstruktor. Beim Lesen holen Sie die Daten über die `ToArray`-Methode. Das einzige Problem ist, die Daten in ein `Byte`-Array zu konvertieren und beim Lesen wieder zurückzukonvertieren. Dieses Problem können Sie für Bilder z. B. über das Codebook lösen (siehe Rezept 323 und 324).

Das folgende Beispiel basiert auf einer WPF-Anwendung, die ein LINQ-to-SQL-Modell für die *Persons*-Datenbank enthält, die der oben angegebenen SQL-Anweisung entspricht. Es erfordert den Import der Namensräume `System`, `System.Data.Linq`, `System.IO`, `System.Linq` und `System.Windows.Media.Imaging`.

```
// DataContext erzeugen
PersonsDataContext dataContext = new PersonsDataContext();

// Neue Person erzeugen
Person newPerson = new Person();
newPerson.FirstName = "Zaphod";
newPerson.LastName = "Beeblebrox";

// Das Bild einlesen
string imageFileName = "C:\\Bilder\\Zaphod.jpg";
BitmapDecoder decoder = BitmapDecoder.Create(new Uri(imageFileName),
   BitmapCreateOptions.None, BitmapCacheOption.OnLoad);
BitmapSource bitmapSource = decoder.Frames[0];

// Das Bild in ein Byte-Array konvertieren (das dem PNG-Format entspricht)
PngBitmapEncoder encoder = new PngBitmapEncoder();
encoder.Frames.Add(BitmapFrame.Create(bitmapSource));
byte[] bitmapData = null;
using (MemoryStream imageStream = new MemoryStream())
{
   encoder.Save(imageStream);
   imageStream.Flush();
   bitmapData = imageStream.ToArray();
}
```

Listing 786: Speichern binärer Daten über LINQ to SQL

```
// Das Bild im Person-Objekt ablegen
newPerson.Picture = new Binary(bitmapData);

// Die neue Person speichern
dataContext.Persons.InsertOnSubmit(newPerson);
dataContext.SubmitChanges();
```

Listing 786: Speichern binärer Daten über LINQ to SQL (Forts.)

Das Lesen binärer Daten erfolgt ähnlich einfach:

```
// DataContext erzeugen
PersonsDataContext dataContext = new PersonsDataContext();

// Die Id der Person definieren
int personId = 1;

// Die Person ermitteln
Person person = dataContext.Persons.Single<Person>(
   p => p.Id == personId);
if (person != null)
{
   // Vorname und Nachname auslesen
   string personName = person.FirstName " " + person.LastName;

   if (person.Picture != null)
   {
      // Die Bilddaten auslesen
      byte[] bitmapData = person.Picture.ToArray();

      // Die Bilddaten in ein BitmapSource-Objekt konvertieren
      BitmapSource bitmapSource = null;
      using (MemoryStream imageStream = new MemoryStream(bitmapData))
      {
         BitmapDecoder decoder = BitmapDecoder.Create(imageStream,
            BitmapCreateOptions.None, BitmapCacheOption.OnLoad);
         bitmapSource = decoder.Frames[0];
      }

      // Das Bild verarbeiten
      ...
   }
}
```

Listing 787: Lesen binärer Daten über LINQ to SQL

ADO.NET

In ADO.NET schreiben Sie in Felder, die binäre Daten erwarten, ebenfalls ein `Byte`-Array. Beim Lesen erhalten Sie auch ein `Byte`-Array zurück, das Sie dann über ein `Stream`-Objekt in eine Datei oder – wenn es sich um ein Bild handelt – in ein `Bitmap`-Objekt schreiben können.

Datensätze mit binären Daten können Sie vereinfacht über ein `DataSet`- oder ein `DataTable`-Objekt hinzufügen. Der Grund dafür liegt darin, dass ein `DataSet`- oder `DataTable`-Objekt die binären Daten automatisch korrekt umsetzt. In einem SQL-Befehl müssten Sie ansonsten erst herausfinden, wie binäre Daten dargestellt werden müssen.

Das folgende Beispiel liest eine JPG-Datei ein und speichert diese mit dem Vor- und dem Nachnamen einer Person in der *Persons*-Tabelle. Es erfordert den Import der Namensräume `System.Data`, `System.Data.SqlClient`, `System.Drawing`, `System.Drawing.Imaging` und `System.IO`.

```
using (SqlConnection connection = new SqlConnection(
   @"Data Source=.\SQLEXPRESS;" +
   @"AttachDbFilename=|DataDirectory|\Persons.mdf;" +
   @"Integrated Security=True;User Instance=True"))
{
   connection.Open();

   // Daten in ein DataTable-Objekt lesen
   SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Persons",
      connection);
   new SqlCommandBuilder(adapter);
   DataTable dataTable = new DataTable();
   adapter.FillSchema(dataTable, SchemaType.Source);

   // DataRow erzeugen und mit den Textdaten füllen
   DataRow newRow = dataTable.NewRow();
   newRow["FirstName"] = "Zaphod";
   newRow["LastName"] = "Beeblebrox";

   // Das Bild einlesen
   string imageFileName = "C:\\Bilder\\Zaphod.jpg";
   Bitmap bitmap = new Bitmap(imageFileName);

   // Das Bitmap in ein Byte-Array umwandeln und speichern
   using (MemoryStream memoryStream = new MemoryStream())
   {
      bitmap.Save(memoryStream, ImageFormat.Jpeg);
      byte[] buffer = memoryStream.ToArray();
      newRow["Picture"] = buffer;
   }
```

Listing 788: Speichern von binären Daten in einer Datenbanktabelle mit ADO.NET

```
   // Datensatz anfügen und Tabelle aktualisieren
   dataTable.Rows.Add(newRow);
   adapter.Update(dataTable);
}
```

Listing 788: Speichern von binären Daten in einer Datenbanktabelle mit ADO.NET (Forts.)

Beim Füllen des `DataTable`-Objekts nutzt das Beispiel übrigens nicht die `Fill`-Methode, sondern `FillSchema`. Damit werden nicht die Datensätze eingelesen (was wir hier nicht benötigen), sondern nur das Schema der Tabelle. Die Performance wird damit erheblich erhöht.

Das Bild wird dann über ein `MemoryStream`-Objekt in ein `Byte`-Array und dieses in die neue Zeile geschrieben. Und schon ist das Bild, nach dem Aufruf der `Update`-Methode des `DataAdapter`-Objekts, in der Datenbank.

Beim Lesen erhalten Sie wieder ein `Byte`-Array, das Sie entsprechend umsetzen müssen. Das folgende Beispiel setzt ein `MemoryStream`-Objekt ein, das mit dem eingelesenen `Byte`-Array initialisiert wird. Über diesen Stream wird dann ein `Bitmap`-Objekt erzeugt.

```
using (SqlConnection connection = new SqlConnection(
   @"Data Source=.\SQLEXPRESS;" +
   @"AttachDbFilename=|DataDirectory|\Persons.mdf;" +
   @"Integrated Security=True;User Instance=True"))
{
   connection.Open();

   // Person einlesen
   int personId = 1;
   string sql = "SELECT * FROM Persons WHERE Id = " + personId;
   SqlCommand command = new SqlCommand(sql, connection);
   using (SqlDataReader reader = command.ExecuteReader())
   {
      if (reader.Read())
      {
         // Das Bild einlesen
         Bitmap bitmap = null;
         byte[] imageData = (byte[])reader["Picture"];
         if (imageData != null)
         {
            using (MemoryStream memoryStream = new MemoryStream(
               imageData, 0, imageData.Length))
            {
               bitmap = new Bitmap(memoryStream);
            }
         }
```

Listing 789: Lesen von binären Daten unter ADO.NET

```
        if (bitmap != null)
        {
           // Das Bild weiterverarbeiten
           ...
         }
       }
   }
}
```

Listing 789: Lesen von binären Daten unter ADO.NET (Forts.)

373 Backup und Restore einer SQL-Server-Datenbank über SMO

Eine SQL-Server-Datenbank können Sie über die Assemblys von SMO (SQL Server Management Objects), die mit dem SQL Server installiert werden, sichern und wiederherstellen. Dies ist besonders interessant für den SQL Server Express, dessen Installation ja leider kein Management-Werkzeug mitliefert. Über SMO können Sie dem Benutzer in Ihren Programmen, die auf den SQL Server (Express) zugreifen, die Sicherung und die Wiederherstellung der verwendeten Datenbank ermöglichen.

Zum Backup oder Restore einer Datenbank definieren Sie zunächst die Verbindungsinformationen zu der Datenbank in einer Instanz der Klasse `ServerConnection` aus dem Namensraum `Microsoft.SqlServer.Management.Common`. Dieser Namensraum versteckt sich entgegen dem sonstigen Vorgehen von Microsoft in der Assembly *Microsoft.SqlServer.ConnectionInfo.dll*.

Die Eigenschaft `ServerInstance` nimmt den Namen des SQL Servers auf. Falls Sie eine vertraute Verbindung zur Datenbank verwenden wollen (also der Benutzer sich mit dem Windows-Login in den SQL Server einloggen soll), setzen Sie die Eigenschaft `LoginSecure` auf `true`. Ansonsten setzen Sie diese Eigenschaft auf `false` und tragen in der Eigenschaft `Login` den Login-Namen und in `Password` das Passwort ein.

Eine Instanz der Klasse `Server` (aus dem Namensraum `Microsoft.SqlServer.Management.Smo`, der in der Assembly *Microsoft.SqlServer.Smo.dll* verwaltet wird) repräsentiert den SQL Server. Wenn Sie diese Instanz erzeugen, übergeben Sie dem Konstruktor das `ServerConnection`-Objekt. Über die Methode `Connect` der `ConnectionContext`-Eigenschaft bauen Sie dann die Verbindung zum Server auf.

Zur Erzeugung eines Backups benötigen Sie ein `Backup`-Objekt. Die Eigenschaft `Action` setzen Sie auf `BackupActionType.Database`, wenn Sie die gesamte Datenbank sichern wollen. Der Wert `Log` ermöglicht das Backup des Transaktionsprotokolls, `Files` ermöglicht das Backup spezieller Datendateien. Die Dokumentation verschweigt aber leider, wie Sie damit umgehen.

Der Eigenschaft `PercentComplete` können Sie eine Instanz des `PercentCompleteEventHandler`-Delegaten zuweisen, der beim Backup (und beim Restore) in regelmäßigen Abständen aufgerufen wird und den aktuellen Fortschritt meldet.

Nun müssen Sie der `Devices`-Eigenschaft über deren `Add`-Methode noch Backup-Geräte (Devices) hinzufügen. Die Objekte, die Sie hier verwenden, sind vom Typ `BackupDeviceItem`.

Dem Konstruktor dieser Klasse übergeben Sie einen Namen und eine Geräte-Typ in Form der `DeviceType`-Aufzählung. Hier können Sie wählen zwischen `File` (Backup in eine Datei), `LogicalDevice` (logisches Backup-Gerät), `Pipe` (Sicherung über das Netzwerk), `Tape` (Sicherung auf Band) und `VirtualDevice` (virtuelles Backup-Gerät). Für die meisten Fälle sollte hier die Sicherung in eine Datei ausreichen. Alle anderen Sicherungsarten werden wohl eher von Administratoren genutzt um produktiv eingesetzte SQL-Server-Datenbanken zu sichern. Wenn Sie `DeviceType.File` wählen, geben Sie am ersten Argument des Konstruktors den (kompletten) Dateinamen an.

Das Backup erstellen Sie dann über die `SqlBackup`-Methode des `Backup`-Objekts, der Sie das zuvor erstellte `Server`-Objekt übergeben. Schließlich trennen Sie die Verbindung über die `Disconnect`-Methode der `ConnectionContext`-Eigenschaft des `Server`-Objekts.

Das Restore einer Datenbank sieht ähnlich aus. Statt einem `Backup`-Objekt verwenden Sie hier ein `Restore`-Objekt, das aber prinzipiell genauso initialisiert wird. Die Eigenschaft `Database` bestimmt den Datenbanknamen. Zum Restore rufen Sie dann die `SqlRestore`-Methode auf.

Listing 790 implementiert eine Klasse mit Methoden zum Backup und Restore in bzw. aus einer Datei. Das Ereignis `PercentComplete` kann dazu genutzt werden, Informationen über den Fortschritt zu erhalten. Die Methode `CreateBackupToFile` erzeugt ein Backup in die angegebene Datei, `RestoreFromFile` stellt eine Datenbank wieder her.

Zum Kompilieren dieser Klasse benötigen Sie Referenzen auf die Assemblys `Microsoft.SqlServer.ConnectionInfo` und `Microsoft.SqlServer.Smo`. Außerdem müssen Sie die Namensräume `Microsoft.SqlServer.Management.Smo` und `Microsoft.SqlServer.Management.Common` importieren.

```
public class SqlServerBackup
{
   /* Wird beim Fortschritt des Backup oder Restore aufgerufen*/
   public event PercentCompleteEventHandler PercentComplete;

   /* Erzeugt ein Backup einer Datenbank */
   public void CreateBackupToFile(string serverName, string databaseName,
      string backupFileName, bool useTrustedConnection, string login,
      string password)
   {
      // Verbindungsinformationen definieren
      ServerConnection serverConnection = new ServerConnection();
      serverConnection.ServerInstance = serverName;
      if (useTrustedConnection)
      {
         serverConnection.LoginSecure = true;
      }
      else
      {
         serverConnection.LoginSecure = false;
         serverConnection.Login = login;
```

Listing 790: Klasse mit Methoden zum Backup und Restore einer SQL-Server-Datenbank

```
        serverConnection.Password = password;
    }

    // Verbindung aufbauen
    Server server = new Server(serverConnection);
    try
    {
        server.ConnectionContext.Connect();

        // Backup in die angegebene Datei erstellen
        Backup backup = new Backup();
        backup.Action = BackupActionType.Files;
        backup.Database = databaseName;
        if (this.PercentComplete != null)
        {
            backup.PercentComplete += this.PercentComplete;
        }
        backup.Devices.Add(new BackupDeviceItem(
            backupFileName, DeviceType.File));
        backup.SqlBackup(server);
    }
    finally
    {
        try
        {
            // Verbindung zum SQL-Server abbauen
            server.ConnectionContext.Disconnect();
        }
        catch { }
    }
}

/* Restauriert ein Backup einer Datenbank */
public void RestoreFromFile(string serverName, string databaseName,
    string backupFileName, bool useTrustedConnection, string login,
    string password)
{
    // Verbindungsinformationen definieren
    ServerConnection serverConnection = new ServerConnection();
    serverConnection.ServerInstance = serverName;
    if (useTrustedConnection)
    {
        serverConnection.LoginSecure = true;
    }
    else
```

Listing 790: Klasse mit Methoden zum Backup und Restore einer SQL-Server-Datenbank (Forts.)

```
      {
         serverConnection.LoginSecure = false;
         serverConnection.Login = login;
         serverConnection.Password = password;
      }

      // Verbindung aufbauen
      Server server = new Server(serverConnection);
      try
      {
         server.ConnectionContext.Connect();

         // Restore aus der angegebenen Datei
         Restore restore = new Restore();
         restore.Action = RestoreActionType.Database;
         restore.Database = databaseName;
         if (this.PercentComplete != null)
         {
            restore.PercentComplete += this.PercentComplete;
         }
         restore.Devices.Add(new BackupDeviceItem(
            backupFileName, DeviceType.File));
         restore.SqlRestore(server);
      }
      finally
      {
         try
         {

            // Verbindung zum SQL-Server abbauen
            server.ConnectionContext.Disconnect();
         }
         catch { }
      }
   }
}
```

Listing 790: Klasse mit Methoden zum Backup und Restore einer SQL-Server-Datenbank (Forts.)

Hinweis

Da Ausnahmen beim Backup und beim Restore nur eine sehr allgemeine Fehlermeldung verwalten, sollten Sie in einer Ausnahme-Behandlung zumindest auch die Nachricht der ersten inneren Ausnahme abfragen, die in der Regel den Grund des Fehlers nennt. Bei der Auswertung der Ausnahmen hilft Ihnen das Rezept 45.

Hinweis

Eine SQL-Server-Datenbank kann nur dann wiederhergestellt werden, wenn zurzeit keine Verbindung zur Datenbank besteht. Falls Sie in einem Programm, das eine Datenbank verwendet, ein Restore dieser Datenbank ausführen wollen, müssen Sie die Verbindung zur Datenbank zuvor schließen (und hoffen, dass kein anderes Programm eine Verbindung zu dieser Datenbank geöffnet hat). Das Problem ist aber, dass ADO.NET einen Verbindungs-Pool verwaltet und die Verbindung intern geöffnet hält. Der Versuch einer Wiederherstellung resultiert innerhalb des Pool-Timeout in einer `Exception` mit der Aussage, dass der exklusive Zugriff auf die Datenbank gerade nicht möglich ist. Um dieses Problem zu lösen müssen Sie den Verbindungs-Pool nach dem Schließen der Verbindung und vor dem Restore leeren:

```
SqlConnection.ClearPool(connection);
```

connection ist hierbei die Referenz auf Ihre Verbindung.

374 Die verfügbaren SQL Server ermitteln

Die verfügbaren SQL Server können Sie (leider nicht ganz zuverlässig) über eine Instanz der Klasse `SqlDataSourceEnumerator` aus dem Namensraum `System.Data.Sql` ermitteln. Über die `Instance`-Eigenschaft dieser Klasse erhalten Sie eine Instanz. Die Methode `GetDataSources` liefert ein `DataTable`-Objekt mit den Daten der ermittelten SQL Server. Das Feld `ServerName` dieser Tabelle speichert den Namen des Servers. Im Feld `InstanceName` wird laut der Dokumentation der Name der Instanz verwaltet, falls diese benannt ist (wie z. B. die Instanz des SQL Server Express). So können Sie relativ einfach alle aufgelisteten SQL Server durchgehen:

```
SqlDataSourceEnumerator enumerator = SqlDataSourceEnumerator.Instance;
DataTable serverTable = enumerator.GetDataSources();
foreach (DataRow row in serverTable.Rows)
{
   string serverName = row["ServerName"].ToString();
   if (row["InstanceName"] != DBNull.Value)
   {
      serverName += "\\" + row["InstanceName"];
   }
   Console.WriteLine(serverName);
}
```

Listing 791: (Unzuverlässiges) Einlesen der aktuell verfügbaren SQL Server

Hinweis

Leider listet `GetDataSources` die verfügbaren SQL Server nicht zuverlässig auf. Die Dokumentation beschreibt schon, dass `GetDataSources` »aufgrund ihrer Natur« nicht immer eine vollständige Liste der verfügbaren SQL Server zurückgibt. Das wesentlichste Problem ist, dass die SQL-Server-Instanzen auf den erreichbaren Rechnern über den SQL-Server-Browser-Dienst abgefragt werden. Läuft dieser nicht, ergibt die Abfrage kein Ergebnis. Sie sollten also wenigstens auf dem lokalen Rechner dafür sorgen, dass dieser Dienst

ausgeführt wird (was Sie über das Rezept 97 erreichen können). Trotzdem wurden auf meinen Systemen keine Express-Instanzen und häufig auch keine der Instanzen des »vollwertigen« SQL Servers aufgelistet.

Eine weitere potenzielle Schwachstelle ist die Tatsache, dass der SQL-Server-Browser-Dienst die Informationen aus der Registry (unter HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server) ermittelt und diese über den Port 1434 per UDP an den Client verteilt. Dabei werden schon einmal versteckte Instanzen ausgenommen (Instanzen können über die SQL-Server-Konfiguration versteckt werden). Ein weiteres Problem kann sein, dass der Port 1434 von einer Firewall blockiert wird.

Sind nach einer Deinstallation eines SQL Servers Reste in der Registry verblieben, kann es auch Fehlinformationen geben. Daneben gibt es Probleme bei gemischten SQL Server 2000/2005 Installationen, die Microsoft in der Knowledgebase unter *support.microsoft.com/kb/922131* beschreibt. SQL-Server-2000-Instanzen müssen mit dem Service Pack 3 versehen sein, damit diese aufgelistet werden können. Na dann ...

375 Die Datenbanken einer SQL-Server-Instanz abfragen

Die Namen der Datenbanken einer SQL-Server-Instanz erhalten Sie über SMO (SQL Server Management Objects). Ähnlich wie in Rezept 373 erzeugen Sie ein `Server`-Objekt (aus dem Namensraum `Microsoft.SqlServer.Management.Smo`), das Sie über ein `ServerConnection`-Objekt initialisieren. Nachdem Sie die Verbindung aufgebaut haben, können Sie die Datenbanken über die `Databases`-Eigenschaft abfragen. Die aufgelisteten `Database`-Objekte bieten eine Vielzahl an Eigenschaften. Über die Eigenschaft `Name` können Sie den Namen der Datenbank auslesen.

Die Methode `GetSqlServerDatabases` in Listing 792 liest auf diese Weise die Datenbanken eines SQL Servers ein, dessen Name im Argument `serverName` übergeben wird. Soll eine vertraute Verbindung (über den Windows-Login des aktuellen Benutzers) verwendet werden, übergeben Sie am zweiten Argument `true`. Im anderen Fall übergeben Sie an den folgenden Argumenten den Login-Namen und das Passwort des SQL-Server-Benutzers.

Zum Kompilieren dieser Methode benötigen Sie Referenzen auf die Assemblys `Microsoft.SqlServer.ConnectionInfo` und `Microsoft.SqlServer.Smo`. Außerdem müssen Sie die Namensräume `Microsoft.SqlServer.Management.Smo` und `Microsoft.SqlServer.Management.Common` importieren.

```
public static ReadOnlyCollection<string> GetSqlServerDatabases(
   string serverName, bool useTrustedConnection, string loginName,
   string password)
{
   // Ergebnis-Liste erzeugen
   List<string> databases = new List<string>();

   // Verbindung zum SQL-Server über SMO aufbauen
   ServerConnection serverConnection = new ServerConnection();
```

Listing 792: Methode zum Auflisten aller Datenbanken einer SQL-Server-Instanz

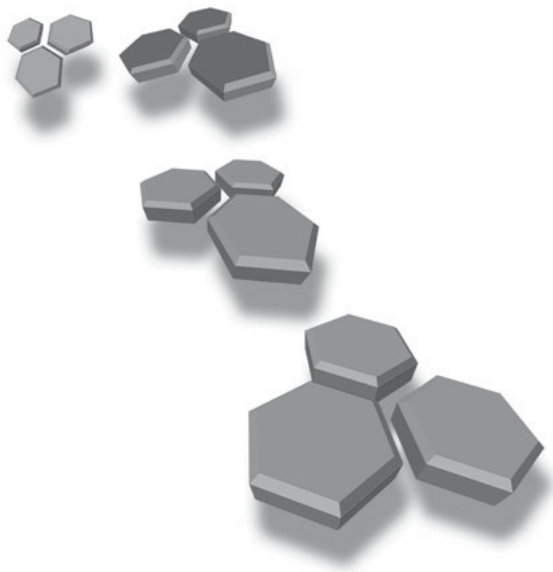
```
   serverConnection.ServerInstance = serverName;
   if (useTrustedConnection)
   {
      serverConnection.LoginSecure = true;
   }
   else
   {
      serverConnection.LoginSecure = false;
      serverConnection.Login = loginName;
      serverConnection.Password = password;
   }
   Server server = new Server(serverConnection);
   try
   {
      // Verbindung aufbauen
      server.ConnectionContext.Connect();

      // Datenbanken abfragen
      foreach (Database database in server.Databases)
      {
         databases.Add(database.Name);
      }
   }
   finally
   {
      try
      {
         // Verbindung zum SQL-Server abbauen
         server.ConnectionContext.Disconnect();
      }
      catch { }
   }

   // Ergebnis zurückgeben
   return new ReadOnlyCollection<string>(databases);
}
```

Listing 792: Methode zum Auflisten aller Datenbanken einer SQL-Server-Instanz (Forts.)

Teil III Anhang



Änderungen gegenüber der zweiten Auflage

Geänderte Rezepte

Basics

45: »Die Nachrichten einer Exception und ihrer inneren Exceptions ermitteln«

Anwendungen, Anwendungs-Konfiguration, Prozesse und Dienste

88: »Externe Konfigurationsdateien lesen und schreiben«: In der vorherigen Auflage habe ich beschrieben, wie Sie in die Konfiguration der aktuellen Anwendung schreiben. Das macht aber eigentlich keinen Sinn mehr, weil dafür die Standard-Einstellungen mit dem Bereich »Benutzer« besser geeignet sind. Deshalb beschreibt dieses Rezept nun, wie Sie die Konfiguration einer anderen Anwendung lesen und schreiben (was z. B. bei einem Setup notwendig sein kann).

102: »Konsolenanwendungen starten und die Ausgabe auswerten«: Dieses Rezept habe ich um die Auswertung des Rückgabecodes und des Fehlerkanals der Anwendung erweitert.

Dateisystem

118: »Die logischen Laufwerke des Systems ermitteln«: Dieses Rezept empfiehlt nun die Verwendung der `GetDrives`-Methode der `DriveInfo`-Klasse an Stelle von `Environment.GetLogicalDrives`.

134: »Programmdateien in den Systempfaden suchen«: Die Methode `FindFileInSystemPaths` berücksichtigte nicht die Tatsache, dass Pfadangaben im Pfad auch in Anführungszeichen eingebettet sein können. Dieser Fall führte zu einer Ausnahme »Illegales Zeichen im Pfad«. Die korrigierte Version berücksichtigt diesen Umstand. Vielen Dank an Mirek Virius von der Czech Technical University, Prague, der mir diesen Fehler meldete.

Text-, binäre und ZIP-Dateien

161: »Komprimieren von Daten mit .NET-Klassen«: Dieses Rezept habe ich um die Klasse `ZipPackage` zur Arbeit mit ZIP-Archiven erweitert.

162: »Dekomprimieren von Daten mit .NET-Klassen«: Dieses Rezept habe ich ebenfalls um die Klasse `ZipPackage` zur Arbeit mit ZIP-Archiven erweitert.

165: »Dateien in ZIP-Archive komprimieren«: Die im vorherigen Codebook beschriebene Methode `ZipFiles` war leider ein klein wenig buggy: Wenn Sie am Argument `comment` `null` übergeben haben, resultierte dies in einer Ausnahme beim Setzen des Kommentars. Deswegen überprüft die verbesserte Version, ob der Kommentar nicht `null` ist.

166: »(ZIP-)Archive aus einem Ordner erzeugen«: Die in diesem Rezept beschriebene Methode `ZipFolder` habe ich um das `params`-Argument `extensionsToInclude` erweitert. An diesem Argument können Sie nun die Endungen der Dateien angeben, die in das Archiv kopiert werden sollen. Ist keine Endung angegeben, werden grundsätzlich alle Dateien übernommen.

168: »(ZIP-)Archive entpacken«: Die in diesem Rezept beschriebene Methode wies zwei Bugs auf: In einigen Fällen speichert ein ZIP-Archiv für die einzelnen Dateien den Unterordner \. In diesem Fall warf die in `ExtractToFolder` verwendete `Path.Combine`-Methode eine `Exception`. Außerdem berücksichtigte `ExtractToFolder` nicht die Tatsache, dass Dateien in ZIP-Archiven auch eine Größe von null Byte aufweisen können. Das Entpacken einer solchen Datei führte ebenfalls zu einer `Exception`.

XML

172: »XML-Dokumente über das X-DOM lesen«: Umgebaut auf das X-DOM

173: »XML-Dokumente beim Einlesen gegen ein Schema (oder DTD) prüfen«: Umgebaut auf das X-DOM

175: »XML-Dokumente validierend über das X-DOM lesen«: Umgebaut auf das X-DOM

176: »Gezielt über XPath auf Elemente in einem XML-Dokument ohne Namensraum zugreifen«: Umgebaut auf das X-DOM

177: »Gezielt über XPath auf Elemente in einem XML-Dokument mit Namensraum zugreifen«: Umgebaut auf das X-DOM

179: »XML-Dokumente über das X-DOM erzeugen und ändern«: Umgebaut auf das X-DOM

183: »Binäre Daten in einem XML-Dokument speichern«: Umgebaut auf das X-DOM

184: »Base64-codierte Bilder (binäre Daten) aus XML-Dokumenten auslesen«: Umgebaut auf das X-DOM

Internet

211: »E-Mails über einen SMTP-Server versenden«: Dieses Rezept habe ich um das Anfordern einer Übertragungs- und einer Lesebestätigung erweitert und zusätzlich die Konfiguration von SMTP in der *app.config* erläutert. Zum Beispiel habe ich außerdem ein Formular hinzugefügt, das Sie in Ihren Projekten als Basis für ein Mailversand-Formular verwenden können.

Windows.Forms

242: »Bei der Betätigung der Return-Taste die Tab-Taste simulieren«: In diesem Rezept ist mir ein kleiner Fehler unterlaufen, der dazu führte, dass das Programm eine `Exception` warf, wenn das Steuerelement, bei dem die Return-Taste in die Betätigung der Tab-Taste umgewandelt wurde, auf einem Container-Steuerelement (z. B. einem `Panel`) lag. In `OnKeyPress` habe ich nicht das ermittelte Container-Steuerelement, sondern den Parent des Steuerelements verwendet, um dessen `ProcessTabKey`-Methode aufzurufen.

244: »Die angezeigten Zeilen einer MultiLine-TextBox auslesen«: In diesem Rezept wurden angezeigte Zeilen nicht korrekt ausgelesen, wenn diese nur aus einem Zeichen bestanden. Diesen von Axel Seibel gefundenen Fehler habe ich korrigiert. Außerdem habe ich den weiteren Fehler mit einer von Axel Seibel gefundenen Lösung korrigiert, der dazu führte, dass Pluszeichen zu Problemen führten.

Sicherheit

309: »Daten symmetrisch ver- und entschlüsseln«: Die in diesem Rezept implementierte Klasse `SymmetricEncryptor` hatte in der Vorversion Probleme mit dem Verschlüsseln von Strings, die Zeichen im Unicode-Bereich über 255 enthielten. Diese Probleme habe ich gelöst. Außerdem habe ich `SymmetricEncryptor` ein wenig umgebaut, sodass jetzt der Schlüssel und der Initialisierungsvektor als Byte-Array oder als String übergeben werden können. Die Klasse enthält zudem nun einige hilfreiche Eigenschaften wie `LegalKeySizes` und `LegalKeySizesString`, über die die möglichen Schlüsselgrößen ausgelesen werden können.

Bildbearbeitung

Die folgenden Rezepte habe ich um die Berücksichtigung von WPF erweitert:

315: »Speicherschonend in einer Windows.Forms-Anwendung mit Bildern umgehen«

318: »Das Format eines Bilds auslesen«

319: »Bild-Metadaten auslesen«

320: »Das Aufnahmedatum eines Bilds auslesen«

321: »Eingelesene Bilder im Originalformat speichern«

322: »Bild in Byte-Array umwandeln«

323: »Byte-Array in Bild umwandeln«

324: »Bilder aus der Zwischenablage auslesen«

325: »Screenshot erstellen«: Dieses Rezept habe ich neben der Berücksichtigung von WPF zusätzlich noch darum verbessert, dass halbtransparente Fenster unterstützt werden.

326: »Bilder skalieren«

327: »Thumbnails aus Bildern erzeugen«

328: »Bilder konvertieren«

329: »(JPEG-)Bilder mit definierter Qualität speichern«

330: »Bilder drehen, neigen und spiegeln«

331: »Bildausschnitte auslesen«

332: »Farben von Bildern auf andere Farben mappen«

333: »Farbinformationen von Bildern gezielt verändern«

334: »Ein Negativ eines Bilds erzeugen«

335: »Die einzelnen Pixel eines Bilds bearbeiten«

336: »Farb-Bilder in Graustufen-Bilder umwandeln«

Zeichnen

Die folgenden Rezepte habe ich um WPF erweitert:

339: »Rechtecke mit abgerundeten Ecken zeichnen«

340: »Pfeile zeichnen«: Dieses Rezept habe ich zusätzlich komplett auf das eigene Zeichnen eines (wesentlich schöneren) Pfeils umgebaut.

341: »Transparente Bilder und Grafiken erzeugen«

342: »Bilder mit Schatten zeichnen«

343: »Schräg zeichnen und Zeichenobjekte rotieren«

344: »Den Drehpunkt eines Rechtecks so ermitteln, dass die Ecke links oben an derselben Position bleibt«

345: »Text an einer definierten Position in 90-Grad-Schritten gedreht ausgeben«

346: »Die Breite und Höhe eines auszugebenden Textes bestimmen«

347: »Texte zentriert oder rechtsbündig«

348: »Strings beim Zeichnen wortgerecht umbrechen«

Reflection und Serialisierung

362: »Objekte nach XML serialisieren und von XML deserialisieren«: In den Methoden zum Serialisieren eines Objekts in einen XML-String und zum Deserialisieren eines XML-Strings in ein Objekt habe ich einen Denkfehler eingebaut. Diesen Methoden konnte die Codierung übergeben werden, was aber beim Serialisieren/Deserialisieren eines (Unicode-)Strings keinen Sinn macht bzw. sogar zu Fehlern führt. Außerdem kam es auch dann zu einem Fehler beim Deserialisieren, wenn beim Serialisieren die Unicode-Codierung angegeben wurde. Der `XmlSerializer` hat in diesem Fall in der im Codebook gedruckten Implementierung der `SerializeToXmlString`-Methode leider ein Byte Order Mark (BOM) Zeichen an den Anfang des Unicode-Zeichen-Stream gehängt. Das hier geschriebene Zeichen 0xFEFF bezeichnet eine UTF-16-Codierung in Big-Endian. Beim Deserialisieren kam es allerdings zu einer Exception (»Ungültiges Zeichen ...«). Der Verzicht auf die Codierung hat die Methoden auch gleich viel schlanker gemacht ☺.

Threading

364: »In einem Thread sicher auf Steuerelemente zugreifen«: Dieses Rezept habe ich um die Berücksichtigung von WPF erweitert.

Datenbank-Programmierung

369: »Datenbanken erzeugen«: Dieses Rezept habe ich um LINQ to SQL erweitert.

370: »Abfragen der automatisch vergebenen Id eines neuen Datensatzes«: Dieses Rezept habe ich ebenfalls um LINQ to SQL erweitert.

371: »Bilder und andere binäre Daten in einer Datenbank verwalten«: Auch dieses Rezept habe ich um LINQ to SQL erweitert.

Neue Rezepte

Basics

43: »Schnelle Auflistung mit Schlüssel- und Indexzugriff«

46: »Exceptions in Anwendungen korrekt auswerten«

Datum und Zeit

53: »Alter in ganzen Jahren berechnen«

75: »Eine Eingabe daraufhin überprüfen, ob diese ein Datum ergeben kann«

76: »Eine Eingabe daraufhin überprüfen, ob diese eine Zeit ergeben kann«

77: »Datumswerte fehlerfrei zwischen Systemen mit verschiedenen Zeitzonen austauschen«

78: »Mit Zeitzonen arbeiten«

79: »Mit Kalendersystemen arbeiten«

Anwendungen, Anwendungs-Konfiguration, Prozesse und Dienste

83: »User Account Control (UAC) berücksichtigen«

89: »Konfigurationsdaten in eigenen Abschnitten speichern«

Dateisystem

113: »Das .NET-Framework-Verzeichnis ermitteln«

XML

169: »XML-Dokumente über LINQ to XML lesen«

170: »XML-Dokumente mit Namensraum über LINQ to XML lesen«

180: »XML-Dokumente über LINQ to XML erzeugen«

185: »XML-Dokumente über LINQ to XML transformieren«

System

205: »System-Hotkeys registrieren und auswerten«

Windows.Forms

230: »Formulare mit dem Vista-Glas-Effekt ausstatten«

WPF

260: »Fenster ohne Titelleiste«

261: »Den Handle eines WPF-Fensters ermitteln«

262: »Fenster über den Clientbereich verschiebbar machen«

263: »Windows-Nachrichten verarbeiten«

264: »Beim Maximieren eines Fensters ohne Titelleiste die Taskbar berücksichtigen«

265: »Fenster verlaufend füllen«

266: »Hintergrund mit Textur«

267: »Fenster mit speziellen Formen«

268: »Fenster mit dem Vista-Glas-Effekt ausstatten«

269: »Fenster in einer Schleife aktualisieren«

270: »Splash-Fenster«

271: »Die aktuelle DPI-Einstellung der Bildschirme des Systems ermitteln«

272: »Ein Fenster auf einem sekundären Bildschirm öffnen«

273: »Das Hauptfenster einer Anwendung ermitteln«

274: »Die absolute und die Bildschirm-Position eines Steuerelements ermitteln«

275: »Die optimale Position eines Fensters bezogen auf ein Steuerelement ermitteln«

276: »Beim Öffnen eines Fensters den Fokus setzen«

277: »Das Einfügen über die Zwischenablage abfangen«

278: »TextBox-Inhalt beim Eintritt komplett selektieren«

279: »TextBox auf Zahleingaben beschränken«

280: »Das TextChanged-Ereignis bei der ComboBox abfangen«

281: »Bei der Betätigung der Return-Taste die Tab-Taste simulieren«

282: »Drag&Drop von Dateien und Ordnern«

283: »In einem Nicht-Tastatur-Ereignis herausfinden, ob eine bestimmte Taste betätigt ist«

LINQ und LINQ to SQL

284: »Dynamische Abfragen«

285: »Ungleichheits-Verknüpfungen«

286: »Kreuzprodukt-Verknüpfungen«

287: »Kommaseparierte Dateien (CSV-Dateien) verarbeiten«

288: »Probleme mit der Benennung in LINQ-to-SQL-Modellen lösen«

289: »LINQ-to-SQL-Abfragen mit LIKE«

290: »SQL direkt ausführen«

291: »Die SQL-Anweisung einer LINQ-Abfrage evaluieren«

Sicherheit

310: »Asymmetrisches Verschlüsseln mit RSA«

311: »Sicherer Schlüsselaustausch mit ECDH«

Bildbearbeitung

316: »Bitmap-Objekte aus BitmapSource-Objekten erzeugen«

317: »BitmapSource-Objekte aus Bitmap-Objekten erzeugen«

COM-Interop mit Office

354: »Performantes Lesen und Schreiben in Excel-Arbeitsmappen«

Serialisierung

360: »Objekte über eine Datenvertrag-Serialisierung serialisieren«

Die ersten 255 Zeichen des Unicode-Zeichensatzes

Abbildung 213 zeigt die ersten 255 Zeichen des Unicode-Zeichensatzes, die mit den Zeichen des ISO-8859-1-Zeichensatzes identisch sind.

	0	1	2	3	4	5	6	7	8	9
0	NUL								BS	HT
10	LF		FF	CR						
20								ESC		
30			SPC	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			
130										
140										
150										
160	NBSP	¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Abbildung 213: Die ersten 255 Zeichen der Unicode-Tabelle

Die grau hinterlegten Zeichen sind Steuerzeichen. Die meisten dieser Zeichen, die den Anfängen der Nachrichtenübertragung entstammen, werden heute nicht mehr verwendet. Tabelle 36 beschreibt die in Abbildung 213 dargestellten, heute noch verwendeten Sonderzeichen.

Sonderzeichen	Wert	Bedeutung
NUL	0	Nullzeichen. Markiert das Ende eines Strings in der Sprache C.
BS	8	Backspace (Rücktaste)
HT	9	Horizontaler Tabulator
LF	10	Line Feed (Zeilenvorschub)
FF	12	Form Feed (Seitenvorschub)
CR	13	Carriage Return (Wagenrücklauf bei der Schreibmaschine)
ESC	27	Escape-Zeichen
SPC	32	Space (Leerzeichen)
NBSP	160	Non Breaking Space (spezielles Leerzeichen, das in HTML bei mehrfachem Vorkommen nicht in ein Leerzeichen umgewandelt wird)

Tabelle 36: Die heute noch verwendeten Sonderzeichen des Unicode- und ISO-8859-1-Zeichensatzes

Hinweis

Informationen über alle Steuer- und sonstigen Sonderzeichen erhalten Sie bei Wikipedia an den Adressen *de.wikipedia.org/wiki/Steuerzeichen*, *de.wikipedia.org/wiki/Unicode-Block_Basis-Lateinisch* und *de.wikipedia.org/wiki/Unicode-Block_Lateinisch-1%2 C _Erg %C3%A4nzung*.

Stichwortverzeichnis

!
#ziplib 401
.NET Framework
  Sourcecode evaluieren 35
.NET Reflector 39

Numerics
8.3-Dateiname 364

A
Abschalten
  des Monitors 527
Absoluter Pfad
  aus relativem 309
  in relativen 307
activeds.tlb 817
AddHook 742
AddNamespace 455
AddPastingHandler 774
AddRange 578
AddressList 552
Administrationsrechte
  prüfen 830
ADO.NET
  Autom. ID abfragen 1114
  Binäre Daten verwalten 1118
  Datenbanken erzeugen 1110
ADSI
  Benutzer ändern 837
  Benutzer anlegen 835
  Benutzer aus Gruppe entf. 841
  Benutzer auslesen 805
  Benutzer löschen 842
  Benutzer zu Gruppe hinzufg. 839
  Benutzereigenschaften 823
  Benutzergruppen 820
  Gruppen anlegen 833
  Gruppen ermitteln 813
  Gruppen löschen 843
  Gruppenbenutzer ermitteln 817
Advanced Security for Outlook 1033
Aes 857
AesCryptoServiceProvider 857
AesManaged 857
Aktivieren
  einer Anwendung 262
Aktualisieren
  Fenster, in Schleife 757
Aktuelle Zeit siehe Systemzeit
AllowDrop 721
AlphaQualityLevel 951
Alphawert 996
Alter
  berechnen 145
Alter berechnen 139
ALTER TABLE 1113
Ancestors 424
Ändern
  Benutzer 837
  des Systemmenüs 637
  von Textdateien 389
  XML-Dateien 467
  XML-Dokumente 460
Andockende Formulare 638
Anfang
  einer Kalenderwoche 165, 166
  einer Woche 153
  eines Monats 154
  eines Quartals 157
Anhalten
  Programm 121
Anlegen
  Benutzer 835
  Benutzergruppe 833
  Datenbanken 1110
Anpassen
  eines DataGridView 703, 707
Anti-Aliasing 984
Anwendungen
  aktivieren 262
  auflisten 271
  Ausführung ermitteln 282

beenden 289
Dateiname ermitteln 215, 299
für asoziierte Dateien starten 284
Hauptfenster ermitteln 768
Hauptformular ermitteln 664
Java-Anw. starten 291
konfigurieren 230, 234, 240, 245
Konsolenanw. starten 287
Mehrfachen Start verhindern 260
Ordner ermitteln 301
Speicherverbrauch reduzieren 226
starten 283
starten und auf Ende warten 285
Verzeichnis ermitteln 216
Anwendungskonfiguration siehe Konfiguration
Anzahl
Datensätze 1109
Kalenderwochen im Jahr 168
Tage im Jahr 148
Tage im Monat 148
Wörter in String 82
AppActivate 262
Application 215, 299, 301
ApplicationName-Eigenschaft 921
ApplyCellStyleToEditingControl 711
AppSettings 235
Ära 208
Arbeitsspeicher
Größe ermitteln 483
überwachen 511, 514, 516
Archive
entpacken 417
erzeugen 409
Informationen auslesen 415
Ordner packen 412
ArcTo-Methode 987
Argumente
Befehlszeile 216
Arrays
durchsuchen 109
sortieren 105
Assemblys
dynamisch erzeugen 1061
dynamisch laden 1062
zur VS-Verw.-Liste 403
AsymmetricAlgorithm 874
Asymmetrische Verschlüsselung
ECDH 878
RSA 873
Asynchrone Methoden
in eigenen Klassen 1092
Asynchrones Entwurfsmuster 1092
AsyncOperation 1094
AsyncOperationManager 1094
Atomuhr-Datum 172
Attachments 558
Attribute 424, 434
Attributes 359, 424
Aufhellen
eines Bilds 966
Auflisten
aller Anwendungen 271
aller Prozesse 269
Auflistungen
durchsuchen 109
Schlüssel und Indexzugriff 120
sortieren 105
Typisierte 115
Aufnahmedatum
eines Bilds 920, 927
Aufrufer
einer Methode 1057
Auftrennen
von Strings 79
Aufzählungen
Konst. als Strings 99
Wert aus String 100
Ausgabe
einer Konsolenanw. ausw. 287
Auslesen
Bildinformationen 920
Ausnahmen
global behandeln 223
Ausnahmen siehe Exceptions
Ausschnitt
eines Bilds 960
Auswerten
System-Hotkeys 529
von Befehlszeilenargumenten 216

Author-Eigenschaft 921
AutoCompleteMode 686
AutoCompleteSource 687
Automatisches Selektieren
  TextBox (Windows.Forms) 668
  TextBox (WPF) 775
Autovervollständigung
  in ComboBox 685
AvailableFreeSpace 316
AVI-Dateien
  in WPF absp. 908
  über MCI absp. 892

B
BackgroundCopyJob 604
BackgroundCopyManager 603
BackgroundWorker 1079, 1082
Backup
  SQL Server 1120
Base64-Codierung 393, 394
BaseUtcOffset 205
Beenden
  Anwendungen 289
  Prozesse 289
Beep ausgeben 71
Befehlszeilenargumente
  auswerten 216
Benutzer
  ändern 837
  anlegen 835
  auf Adminrechte prüfen 830
  auf Gruppenzugehörigkeit prüfen 830
  aus einer Gruppe entf. 841
  auslesen 805
  Eigenschaften lesen 823
  einem Progr. zuweisen 845
  einer Gruppe 817
  einer Gruppe hinzufg. 839
  Existenz prüfen 811
  Gruppen abfragen 820
  löschen 842
  Name des aktuellen 844
Benutzergruppen siehe Gruppen
Berechnen
  Alter in Jahren 145
Betätigung
  der Return-Taste abfragen 675
  einer Taste abfragen 723, 786
  von Tasten simulieren 296
Bilder
  Aufnahmedatum 920
  aus Byte-Array 934
  aus d. Clipboard 936
  Ausschnitte 960
  drehen 955
  einzelne Pixel ändern 973
  Erzeugungsdatum auslesen 927
  Farben verändern 962, 966
  Format auslesen 915
  Graustufen 979
  Helligkeit verändern 966
  in Byte-Array 932
  in einer Datenbank 1115
  Informationen auslesen 920
  konvertieren 949, 951
  mit Schatten 1001
  Negativ 971, 973
  Qualität beim Speichern 951
  skalieren 943
  speicherschonend bearbeiten 911
  spiegeln 955
  Thumbnails 947
  Transparente 996
Bildformat
  ermitteln 915
Bildschirm
  Fenster auf sekundärem 766
Bildschirm siehe Monitor
Bildschirm-Position
  eines Steuerelements 659, 768
Binäre Dateien
  aus Base64-Strings 394
  Base64-codieren 393
  lesen 390
  schreiben 392
Binäre Daten
  aus XML lesen 470
  in einer Datenbank 1115
  in XML 469
Binary 1116

BinaryFormatter 1071
BinarySearch 112
Binärzahlen siehe Dualzahlen
BitBlt 939
Bitmap
  nach BitmapSource 914
  von BitmapSource 913
BitmapEffect 1002
BitmapMetadata-Klasse 921
Bitmaps siehe Bilder
BitmapSource
  nach Bitmap 913
  von Bitmap 914
BITS 601
BLOB 1115
Bogenmaß
  in Gradmaß 55
Booten
  des Computers 503
Breite eines Textes 1015
Browser
  in Anw. integrieren 563
  starten 562
Bubblesort 701
Buddhistischer Kalender 209
Byte-Arrays
  aus binärer Datei 390
  aus Strings erz. 101
  binäre Datei erz. 392
  in Bitmap umw. 934
  in String umw. 104
Byte-Werte
  lesbares Format 60

C

Calendar 148
CalendarWeekRule 161
CameraManufacturer-Eigenschaft 921
CameraModel-Eigenschaft 921
CancelAsync 1083
CancelCommand 774
CanPauseAndContinue 278
CBT-Hook 728
CellTemplate 716
CenterX 955
CenterY 955
CharSet 321
Children 833
ChineseLunisolarCalendar 208
Chinesischer Kalender 208
ClassDefNotFoundError 293
Classpath 292
Clipboard 774
Clipboard siehe Zwischenablage
Clipboard-Klasse (WPF) 936
CngAlgorithm 878
Code siehe Programmcode
CodeDomProvider 1058
Codierung 101
Collections siehe Auflistungen
Color-Eigenschaft 1002
ColorMap 964
ColorMatrix 967
ComboBox
  Auto-Complete 685
  Zusätzl. Daten 692
ComboBox (WPF) 781
COM-Komponenten
  laufende Instanzen ref. 1031
Comment-Eigenschaft 921
CommitChanges 833, 835
CompareTo 105, 185
Comparison 106
CompositingMode 985
CompositingQuality 985
CompositionTarget 765
ComputeHash 882
Computer
  booten 503
  Name ermitteln 477
config-Datei
  eigene Handler 240
  lesen 234, 240
  lesen und schreiben 230, 235
Configuration 236
ConfigurationManager 235
ConnectionStrings 236
ConnectionStringsSection 236
Continue 276
Convert 394

Copy 349
CopyFromScreen-Methode 938
CopyPixelOperation-Aufzählung 938
CopyPixels-Methode 932
Copyright-Eigenschaft 921
CounterCreationData 522
CounterCreationDataCollection 522
CPU
  Geschwindigkeit 482
Create 240, 880
CREATE DATABASE 1111
CREATE TABLE 1112
CreateBitmapSourceFromHBitmap 914
CreateDecryptor 869
CreateDirectory 318
CreateEncryptor 868
CreateOperation 1094
CreateSpecificCulture 56, 133
CreateSubKey 252, 255
Credentials 556
CroppedBitmap 960
Cross Joins 794
CryptographicException 876, 877
Cryptography Next Generation 880
CryptoStream 868
CSV-Dateien
  mit LINQ einlesen 794
CultureInfo 56, 133, 148
CurrentCulture 56
CurrentDirectory 293
CurrentThread 56
Cursorposition
  in einer TextBox ermitteln 678

D
DataContract-Attribut 1067
DataContractSerializer-Klasse 1066
DataGridView
  anpassen 703, 707
  Eigene Spalten 707
  mit DateTimePicker 707
  Zellen lesen/schreiben 702
DataGridViewCell 708
DataGridViewTextBoxCell 708
DataMember-Attribut 1067
DataObject 774
DataObjectPastingEventHandler 774
DataSet 465
DataSetName 465
DataTable 465
DataTaken-Eigenschaft 921
DateAndTime 139
DateDiff 139
Dateiattribute
  lesen 359
  setzen/aufheben 361
Dateidatum
  lesen/ändern 362
Dateien
  8.3-Name ermitteln 364
  Attribute aufheben 361
  Attribute lesen 359
  Attribute setzen 361
  auf FTP-Server löschen 631
  aus (ZIP-)Archiven 417
  Base64-codierte 393, 394
  binäre Daten aus XML 470
  binäre Daten in XML 469
  Binäre lesen 390
  Binäre schreiben 392
  CSV-Dateien mit LINQ einlesen 794
  Datum lesen/ändern 362
  Drag&Drop 720, 785
  eines Ordners aufl. 366
  Endung ändern 307
  Existenz prüfen 344
  Größe ermitteln 358
  Größe von FTP-Server abfragen 612
  in Papierkorb versch. 371
  in Sys.-Pfaden suchen 346
  komprimieren 409
  kopieren 349
  löschen 352
  mit assoziierter Anw. öffnen 284
  mit BITS downloaden 601
  suchen 344
  Textdateien ändern 389
  Textdateien lesen 383
  Textdateien schreiben 385
  Texte anfügen 387

überwachen 380
umbenennen 351, 367
vergleichen 353
Verknüpfungen 377
verschieben 351, 367
Version auslesen 363
von einem FTP-Server downloaden 569
von einem Webserver downloaden 569, 578, 601
XML transformieren 473
XML-Dateien ändern 467
XML-Dateien lesen 433, 451
XML-Dateien schreiben 457, 460, 465
zu einem FTP-Server uploaden 614
zu einem Webserver uploaden 601, 614
Zugriffsdaten lesen/ändern 362
Dateigröße
von FTP-Server abfragen 612
Dateiname
der Anwendung 215
einer Klassenbibliothek 300
einer Windows-Anw. 299
für temporäre Dateien 302
Dateisystem
überwachen 380
Daten
dekomprimieren 399, 407
entschlüsseln 855, 873
Hashes erzeugen 879
komprimieren 395, 401
verschlüsseln 855, 873
zus. in Listbox 692
Datenbanken
Anzahl Datensätze 1109
Backup und Restore 1120
Bilder verwalten 1115
Binäre Daten 1115
eines SQL Servers abfragen 1125
erzeugen 1110
Datensätze
autom. Id abfragen 1113
Datenvertrag-Serialisierung 1066
DateTime 148, 160
Datumstyp 202
DateTimeFormat 150
DateTimeFormatInfo 150
DateTimeKind 202
DateTimeOffset 203
DateTimePicker
in DataGridView 707
Datum
addieren, subtr. 145
Aktuelles 131
Differenz 138
einer Datei 362
Eingabe validieren 197
Feiertag? 192
formatieren 131
Gültigkeit 65
in String konvertieren 212
ISO-Datum 136
Kalendersysteme 207
Kalenderwoche 161
Länderformate 133
Monatsstart und -ende 154
normalisieren 177
Quartal berechnen 157
Quartalstart und -ende 157
Systemdatum setzen 169
vergleichen 137
von Atomuhr 172
Wochenstart und -ende 153
Wochentag ermitteln 149
zwischen Kalendersystemen konvertieren 212
zwischen Systemen übertragen 202
Datumstyp 202
DaylightName 205
DayOfWeek 149, 161
DaysInMonth 148
Debuggen
.NET-Framework-Sourcecode 35
Decrement 525
Decrypt 877
DefaultNewRowValue 714
Deflate 399
Deflate-Algorithmus 395
DeflateStream 395, 399
Dekomprimieren
von ZIP-Archiven 417

ZIP und andere 407
ZIP, Deflate, GZIP 399
Delete 256, 352
DeleteMenu 637
DeleteSubKey 257
DeliveryMethod 556
DeliveryNotificationOptions 558
DeriveKeyMaterialMethode 878
DES 857
Descendants 424
DESCryptoServiceProvider 857
Deserialisieren 1071
Datenvertrag-Serialisierung 1069
Deserialize 1071
Desktopgestaltung 753
Device Context 939
Dictionaries
Typisierte 115
Dictionary 115
Dienste
anhalten 276
Informationen auslesen 272
starten 276
stoppen 276
Differenz
Datumswerte 138
Zeitzonen 206
Digitale Signaturen 873
Direction-Eigenschaft 1002
Directory 317, 318, 336
DirectoryEntry 807, 817, 833, 835, 842
DirectoryInfo 317, 338, 366
Dispatcher 1080
DispatcherUnhandledException-Ereignis 224
DisplayName 205
Dispose 227
DoEvents 757
Dokumentenvorlagen
verwenden 1037
DOM 460
Domäne
Benutzer auslesen 805
Gruppen auslesen 813
DownloadData 570
DownloadDataAsync 570
DownloadFile 570
DownloadFileAsync 570
DownloadString 570
DownloadStringAsync 570
DPAPI 851
DPI 766
Systemeinstellung ermitteln 765
Drag&Drop
Dateien & Ordner 720, 785
unter UAC 220
DragDrop 721
DragEnter 721
DragMove 741
Drehen
einer Zeichnung 957
von Bildern 955
Drehpunkt
eines Rechtecks 1011
DriveInfo 313, 316
DropShadowBitmapEffect 1002
DSA 874
DSACryptoServiceProvider 874
Duale Darstellung 46
Dualzahlen
aus Dezimalzahlen 46
konvertieren 51
Durchsuchen
Arrays 109
Auflistungen 109
DwmExtendFrameIntoClientArea 647, 753
DwmIsCompositionEnabled 647, 753
Dynamische Abfragen (LINQ) 791
Dynamisches Ausführen
von Programmcode 1058
Dynamisches Erzeugen
von Assemblys 1061
Dynamisches Laden
von Assemblys 1062

E

ECDH 878
ECDiffieHellman 874
ECDiffieHellmanCng 874, 878

ECDiffieHellmanKeyDerivationFunction-Aufzählung 878
ECDsa 874
ECDsaCng 874
EditingControlDataGridView 709
EditingControlFormattedValue 709
EditingControlRowIndex 710
EditingControlValueChanged 710
EditingControlWantsInputKey 711
EditingPanelCursor 710
EditType 714
Einfügen
über die Zwischenablage abfangen 774
Eingabefokus siehe Fokus
Eingaben
auf Datum prüfen 197
auf Zeit prüfen 201
Elapsed 195
ElapsedMilliseconds 195
ElapsedTicks 195
Element 434
Elements 434
ElementsAfterSelf 424
ElementsBeforeSelf 424
Elliptic Curve Diffie-Hellman-Algorithmus 878
E-Mail
Empfangsbestätigung 558
Lesebestätigung 558
Übertragungsbestätigung 558
E-Mails
über MAPI vers. 560
über Outlook vers. 560
über Pickup senden 556
über SMTP vers. 554
Empfangsbestätigung
bei SMTP-E-Mail 558
EnablePrivileges 503
EncoderParameters 953
Encoding 101, 383
Ende
einer Woche 153
eines Monats 154
eines Quartals 157
Energiesparmodus
Monitor 527
Entpacken siehe Dekomprimieren
Entwurfsmuster
Asynchrones 1092
Enum 99
Enumerationen siehe Aufzählungen
Environment 303, 347, 477
Environment-Klasse 217
Ereignisbasiertes asynchrones Entwurfsmuster 1092
Ereignisse
OnThreadException 224
Error-Ereignis 380
Ersetzen
Strings am Anfang 91
von Strings 72
Erzeugen
Binäre Dateien 392
Datenbanken 1110
Textdateien 385
von Ordnern 318
XML-Dokumente 457, 460, 465
XML-Dokumente über LINQ to XML 463
Erzeugungsdatum
eines Bilds 927
Escape 91
Evaluieren
.NET-Framework-Sourcecode 35
Excel-Arbeitsmappen
erweitern 1047
erzeugen 1041
Exceptions
Alle Nachr. auslesen 122
Exceptions siehe Ausnahmen
ExecutablePath 299
ExecuteQuery 799
EXIF-Standard 920
Existenz
einer Datei 344
eines Benutzers prüfen 811
eines Ordners 317
Exists 317, 344
Express ClickYes 562

F
Farben
eines Bilds ändern 962, 966
Feiertage
berechnen 181
Datum = Feiertag? 192
im MonthCalendar darst. 717
Fenster (WPF)
auf sekundärem Bildschirm 766
Fokus auf das erste Steuerelement 774
Handle ermitteln 740
Hauptfenster einer Anwendung 768
Hintergrund mit Textur 748
in Schleife aktualisieren 757
mit speziellen Formen 751
ohne Titelleiste 740
ohne Titelleiste maximieren 743
optimale Position 770
Splash-Fenster 761
Titellose bewegen 741
über Clientbereich verschieben 741
verlaufend füllen 747
Vista-Glas-Effekt 753
File 344, 349, 351, 383
FileInfo 344, 358, 359, 383, 390
FileStream 390, 392
FileSystemWatcher 380
FileVersionInfo 363
FindWindow 263
Finish 405
FixedDocument 1033
Fokus
auf das erste Steuerelement 774
Fokuswechsel
durch Cursortasten verhindern 725
FolderBrowserDialog 722
Folders 1054
Font-Objekte
serialisieren 1076
Format 131, 133
eines Bilds ermitteln 915
FormatConvertedBitmap 980
Format-Eigenschaft 921
Formatieren
Datum 131
von Zahlen 43, 45
FormattedText 1018, 1022
FormBorderStyle 635
Formulare
Andockende 638
auf sek. Bildschirm 658
Hauptformular einer Anwendung 664
Monitor ermitteln 654
ohne Titelleiste 635
optimale Position 660
Rahmenlose bewegen 635
Schließen-Schalter entf. 639
Screenshot 938
Spezielle Formen 641
Splash-Forms 652
Sys-Menü ändern 639
über Clientbereich versch. 635
Unbewegbare 637
verlaufend füllen 640
Formulare (WPF)
Vista-Glas-Effekt 647
Freier Speicherplatz
eines Laufwerks 316
FromBase64String 394, 470
FromHwnd 742
FromXmlString 876
FTP
Dateien löschen 631
Dateigröße abfragen 612
Download 569
Ordner erstellen 632
Ordner löschen 631
Upload 614
Verzeichnisse lesen 621
Funktionale Konstruktion 462

G
Garbage Collector
Collection erzwingen 228
GC.Collect 228
GDI
Probleme vermeiden 983
Gemeinsame Menge 115

GenericTypographic 1016
Geschützte Verbindung 576, 600
Geschwindigkeit
  des Prozessors 482
GetActiveObject 1031
GetAsyncKeyState 723
GetAttribute 430
GetBytes 101
GetCategories 506
GetCommandLineArgs-Methode 217
GetCounters 507
GetData 937
GetDataPresent 721
GetDaysInYear 148
GetDirectories 338
GetDrives 311
GetEditingControlFormattedValue 711
GetEncoding 383
GetEntryAssembly 215, 299
GetEnvironmentVariable 347
GetExecutingAssembly 299, 300
GetFiles 338
GetFolderPath 303, 477
GetFrame 1057
GetHbitmap 914
GetHdc 940
GetHostByName 552
GetImage-Methode 936
GetInstanceNames 507
GetLastWin32Error 265
GetNames 99, 100
GetNextEntry 415
GetNonZeroBytes 98
GetPart-Methode 400
GetParts 401
GetProcesses 269, 271
GetProcessesByName 260, 282, 289
GetResponse 578
GetResponseStream 578
GetShortPathName 364
GetString 104
GetSubKeyNames 488
GetTempFileName 302
GetTempPath 303
GetThumbnailImage 948
GetUtcOffset 205
GetValue 252
GetVersionInfo 363
GetWeekOfYear 161
GetWindowsDirectory 303, 477
GetWindowText 265
GetWindowTextLength 265
GifBitmapEncoder 952
Glas-Effekt 647, 753
Globales Behandeln von Ausnahmen 223
GNU General Public License 402
GoBack 566
GoForward 566
GoHome 566
Google
  Newsgroup-Recherche 30
GPL-Lizenz 402
GradiantStop 747
Gradmaß
  in Bogenmaß 55
Grafiken
  Transparente 996
  verlaufend füllen 640
Grafikobjekte
  transformieren 1004
Graphics 945, 957, 964, 969
GraphicsPath 644, 989
Graustufen 979
GregorianCalendar 208
Gregorianischer Kalender 207, 208
Größe
  aller Unterordner 340
  des Arbeitsspeichers 483
  einer Datei 358
  einer Datei (FTP) 612
  eines Laufwerks 316
  eines Ordners 338
Gruppen
  anlegen 833
  auslesen 813
  Benutzer entfernen 841
  Benutzer ermitteln 817
  Benutzer hinzufügen 839
  eines Benutzers 820
  löschen 843

Guid 895
Gültigkeit
  Datumseingabe 65
  von XML prüfen 441
  von Zahlen überpr. 61
GZIP 399
GZIP-Algorithmus 395
GZipStream 395, 399

H
Handle
  eines WPF-Fensters 740
Handler
  für Konfig.-Daten 240
HashAlgorithm 878
Hashcodes 879
Hashing-Verfahren 879
HashSet 115
Hashtables
  Typisierte 115
HasSameRules 205
Hauptfenster
  einer Anwendung ermitteln 768
Hauptformular
  einer Anwendung ermitteln 664
Hauptversion
  von Windows 480
Hebräischer Kalender 208
HebrewCalendar 208
Helligkeit
  eines Bilds verändern 966
Herunterfahren
  des Computers 503
Herunterladen siehe Download
Hexadezimale Darstellung 45
Hexadezimalwerte
  konvertieren 46
High-Word lesen 69
High-Word setzen 70
HijriCalendar 208
Hijri-Kalender 208
Hintergrund
  mit Textur 748
  transparenter 642
HMAC 881
HmacKey 878
HMACMD5 881
HMACRIPEMD160 881
HMACSHA1 881
HMACSHA256 881
HMACSHA384 881
HMACSHA512 881
Hochladen siehe Upload
Höhe eines Textes 1015
Hooking 727
Hotkeys auswerten 529
HTML-Dokumente
  in Anw. darstellen 563
  in Browser öffnen 562
HTTP
  Download 569, 578
  Upload 614
https 576, 600
HttpUnhandledException
  auswerten 123
HwndSource 742

I
IADsGroup 817
IADsUser 818, 820, 821
ICMP
  Prüfsumme berechnen 546
IComparable 105, 107, 185
IComparer 107
IConfigurationSectionHandler 240
Icons
  aus Byte-Array 934
  in Byte-Array 932
ICryptoTransform 868
IDataGridViewEditingControl 708
IFormatProvider 60
ImageAttributes 964
ImageBrush 748
ImageCodecInfo 953
ImageMetadata-Klasse 921
ImageQualityLevel 951
Imaging 914
Impersonation 845
Increment 525
IndexOf 110

Informationen
aus Bildern auslesen 920
Initialisierungsvektor 856
InitializeEditingControl 714
InnerExeption 122
InputManager 783
Installierte Programme 488
InstanceName 515
Interlaced 951
Internet
Verb. öffnen/schließen 548
Verbindung prüfen 544
InternetAutodial 548
InternetAutodialHangup 548
InternetGetConnectedStateEx 541, 544
Internetverbindung
öffnen und schließen 548
Status abfragen 541
InterpolationMode 945, 985
Invoke 835, 839, 1080, 1081
InvokeRequired 1081
IP-Adressen
ermitteln 552, 553
IPHostEntry 552
Iranischer Kalender 209
IsBodyHtml 557
IsDaylightSavingTime 205
IShellLink 378
IsIconic 263
Islamischer Kalender 208
IsLeapYear 160
ISO-8601 136, 161
ISO-8859-1-Zeichensatz 1136
ISO-Datum 136
IV 856

J
Jahr
Anzahl Tage 148
Schaltjahr 160
JapaneseCalendar 209
JapaneseLunisolarCalendar 209
Japanischer Kalender 209
java 292
Java-Anw. starten 291
Java-Interpreter 292
javaw 292
JPEG-Bilder
Qualität 951
JpegBitmapEncoder 951
Jüdischer Kalender 208
JulianCalendar 208
Julianischer Kalender 208

K
Kalender
Ära 208
Buddhistischer 209
Chinesischer 208
Datumswerte konvertieren 212
Gregorianischer 207, 208
Hebräischer 208
Hijri 208
Iranischer 209
Islamischer 208
Japanischer 209
Jüdischer 208
Julianischer 208
Koreanischer 209
Lunarkalender 207
Lunisolarkalender 208
Persischer 209
Saudiarabisch islamischer 210
Solarkalender 207
Taiwanesischer 210
Thailändischer 209
Umm Al-Qura 210
Kalendersysteme 207
Kalenderwoche
eines Datums 161
Startdatum 165, 166
Kantenglätung 984
KeyDerivationFunction 878
KeyedCollection 120
Keywords-Eigenschaft 921
Kill 290
Klassen
mit asynchronen Methoden 1092
Klassenbibliothek
Dateiname ermitteln 300

Ordner ermitteln 302
Knowledge-Base-Recherche 34
Kommaseparierte Dateien
mit LINQ einlesen 794
Komprimieren
Ordner und Dateien 409
ZIP (und andere) 401
ZIP, Deflate, GZIP 395
Konfigurationsdaten
aus .config lesen 240
Externe lesen und schreiben 235
in .config speichern 230
in XML-Datei speichern 245
Konfigurieren
Anwendung 230, 234
über eigene Klasse 245
Konsolenanwendungen
starten und Ausgabe ausw. 287
Kontakte
aus Outlook 1053
Konvertieren
Kalender-Datumswerte 212
nach JPEG 951
von Bildern 949
von Dualzahlen 51
von Hexadezimalzahlen 46
Zeit in DateTime 147
Kopieren
eines Ordners 318, 321
Ordner ohne Abbruch 328
von Dateien 349
KoreanCalendar 209
Koreanischer Kalender 209
KoreanLunisolarCalendar 209
Kreuzprodukt-Verknüpfungen 794
Krüger, Mike 402
Kultur umstellen 56
Kulturen
Datumswerte 133
verfügbare auslesen 57
Zugeordnete Kalender 210
Kürzen
von Strings 89

L
Language Integrated Query siehe LINQ
Laufwerke
eines best. Typs erm. 314
ermitteln 311
Freier Speicherplatz 316
Größe ermitteln 316
Typ ermitteln 313
LayoutTransform 1005
Leerzeichen
in ein Leerz. 80
LegalBlockSizes 860
Leistungsindikatoren 505
auflisten 506
Eigene implementieren 522
Speicher 511
Speicher und CPU 514
Length 358
Lesebestätigung
bei SMTP-E-Mail 558
Lesen
aus der Registry 252
Binäre Dateien 390
Konfigurationsdaten 230, 234, 245
von Konfig.-Daten 240
von Textdateien 383
XML-Dokumente 423, 429, 433, 451
LIKE
in LINQ to SQL 798
LinearGradientBrush 747
LineTo-Methode 987
LinkedList 115
LINQ
CSV-Dateien einlesen 794
Dynamische Abfragen 791
Kreuzprodukt-Verknüpfungen 794
Progressive Abfragen 790
Ungleichheits-Verknüpfungen 793
LINQ to SQL
Autom. ID abfragen 1114
Binäre Daten verwalten 1116
Datenbanken erzeugen 1110
Falsche Benennungen korrigieren 796
LIKE-Operator 798

SQL direkt ausführen 799
SQL evaluieren 800
LINQ to XML
XML-Dokumente erzeugen 460, 463
XML-Dokumente lesen 423, 433
XML-Dokumente transformieren 471
LINQ To XML siehe LINQ
List 115
ListBox
mit Tooltip 693
Zusätzl. Daten 692
ListView
sortieren 695
Load 451
LoadXml 249
localhost 555
LockBits 977
Logische Laufwerke
ermitteln 311
LogonUser 845
Löschen
Benutzer 842
Benutzergruppe 843
Dateien auf FTP-Server 631
in den Papierkorb 371
Ordner auf FTP-Server 631
von Dateien 352
von Ordnern 337
Lossless 952
Low-Word lesen 69
Low-Word setzen 70
Lunarkalender 207
Lunisolarkalender 208
Lutz Roeder 39

M

MachineName 477
MACTripleDES 881
mail.gmx.de 556
MailAttachment 558
MailMessage 557
Mails siehe E-Mails
MainWindowTitle 271
MakeTransparent 999
Mallen, Ronald W. 182
ManagementObject 482, 483, 503
ManagementScope 503
Manifest
für UAC 221
MAPI 560
E-Mails vers. 560
Mapping
von Farben 962
Marshal 265, 1031
MarshalAs 323
MaskedTextBox
Cursorposition ermitteln 678
Matches 82, 94
Math 54
Matrix 957, 1008
Matrix-Klasse 1010
MatrixTransform-Klasse 1005
Mattias Sjögren 378
Maximieren
eines Fensters ohne Titelleiste 743
MaxTextHeight 1018, 1022
MaxTextWidth 1018, 1022
MaxValue 55
MCI 892
mciGetErrorString 893
mciSendCommand 892
MD5 881
MD5CryptoServiceProvider 881
MeasureString 1016
MediaElement 908
Mehfachstart verhindern 260
Mengenoperationen 115
MenuItems 665
Menüs
Laufzeit 665
Message Digest 881
MessageBox
Schalterbeschr. ändern 727
Messen
Zeit 194
Metafile
Aus Byte-Array 934
In Byte-Array 932
Methoden
Aufrufer ermitteln 1057

Methodenaufrufe
umleiten 1080, 1081
Microsoft Standard User Analyzer 222
Midi-Dateien
in WPF absp. 908
über MCI absp. 892
MINMAXINFO 743
MinValue 55
Missing.Value 1034
ModifierKeys 723
Monat
Anzahl Tage 148
Name ermitteln 152
Start und Ende berechnen 154
Monitor
abschalten 527
eines Formulars 654
einschalten 527
Energiesparmodus 527
Fenster auf sekundärem 766
Formular auf sekundärem 658
Screenshot 938
MONITOR_DEFAULTTONEAREST 744
MonitorFromWindow 744
MONITORINFO 744
MonthCalendar
Feiertage darst. 717
Move 336, 351
MoveToContent 430
MP3
in WPF absp. 908
über MCI absp. 892
MPEG-Dateien
in WPF absp. 908
über MCI absp. 892
MPEGVideo-Treiber 895
MSDN
Recherche 33
Multimediadateien absp. 892, 908
Mutex 260

N
Nachrichten
aus Exception lesen 122
Name
Aktueller Benutzer 844
des Computers 477
eines Kalendersystems 211
eines Monats 152
eines Wochentags 150
Namensräume
XML 427
Namespace 465
Negativ
eines Bilds 971
NetDataContractSerializer-Klasse 1066
NetworkCredential 556
NewGuid 895
Newsgroup-Recherche 30
NIST 172
Noise-Eigenschaft 1002
Non-Equi Join 793
Normalisieren
eines Datums 177
Notationen
von Zahlen 52
NumberStyles 63

O
Objekte
binär deserialisieren 1071
binär serialisieren 1071
nach XML serialisieren 1073
per Datenvertrag deserialisieren 1066
per Datenvertrag serialisieren 1066
von XML deserialisieren 1073
Öffentlicher Schlüssel 873
Office
Excel-Arbeitsmappen 1041, 1047
Excel-Dat. erweitern 1047
Excel-Mappen erz. 1041
Outlook-Kontakte 1053
Word-Dokumente 1033, 1037
Word-Dokumente erz. 1037
Word-Dokumente öffnen 1033
Offset
einer Zeitzone 206
OnThreadException-Ereignis 224
Opacity-Eigenschaft 1002

OpenExeConfiguration 236
OpenIcon 263
OpenRead 390
OpenSubKey 252
OpenText 383
OpenWrite 392
Optimale Position
  eines Fensters 770
  eines Formulars 660
Ordner
  archivieren 412
  auf FTP-Server erstellen 632
  auf FTP-Server löschen 631
  Dateien auflisten 366
  der Anwendung 216
  Dialog 722
  Drag&Drop 720, 785
  durchsuchen 344
  einer Klassenbib. 302
  einer Windows-Anw. 301
  erzeugen 318
  Existenz prüfen 317
  für temp. Dateien 303
  Größe aller Unterordner 340
  Größe ermitteln 338
  in Archiv packen 412
  in Papierkorb versch. 371
  komprimieren 409, 412
  kopieren 318, 321
  löschen 337
  ohne Abbruch kopieren 328
  relative ermitteln 307
  umbenennen 336, 367
  vergleichen 355
  Verknüpfungen 377
  verschieben 337, 367
  Windows-Ordner 303
Ostern berechnen 182
Outlook
  E-Mails versenden 560
  Kontakte auslesen 1053

P
Packen siehe Komprimieren
Padding 856
Papierkorb
  Anzahl der Dateien erm. 374
  Dateien verschieben 371
  Größe ermitteln 374
Parallele Schnittstellen
  auflisten 491
Param 953
Parse 178
PartExists-Methode 400
Passwort
  per Zufall berechnen 97
Passwörter
  verschlüsseln 855, 879
Path 302, 303
PathGeometry 987
PathRelativePathTo 307
Path-Umgebungsvariable 346
Pattern siehe Entwurfsmuster
Pause 276
Performance
  bei der Suche in einer Auflistung 111
  BinarySearch 114
Performance Counter siehe Leistungsindikatoren
PerformanceCounter 506, 512, 515, 524
PerformanceCounterCategory 506, 522
PersianCalendar 209
Persischer Kalender 209
Personifizierung 845
Pfadangaben
  überprüfen 310
Pfade
  relative ermitteln 307
  relative in absolute 309
Pfeile zeichnen 991
Pickup-Auslieferung 556
Pickup-Verzeichnis 556
Piepston ausgeben 71
Ping 544
PingReply 545
Pixel
  Einzelne bearbeiten 973
  transformieren 966, 981
Platform 480
Play 891

PlayLooping 891
PlaySync 891
Plug-In-System 1063
PngBitmapEncoder 951
PointToScreen 769
Pop-Methode 1006
Position
  des Eingabecursors einer TextBox 678
  relativ zum Bildschirm 659, 768
PPI 766
Prefix 465
PrepareEditingControlForEdit 712
PresentationSource 765
PreviewTextInput 776
Priority 558
Privater Schlüssel 873
Probleme
  mit GDI vermeiden 983
Process 269, 271, 283, 284, 289, 562
ProcessInput 783
ProcessName 271, 515
ProcessTabKey 676
Programmcode
  dynamisch ausführen 1058
Programme
  anhalten 121
  Installierte auflisten 488
  Personifizierung 845
ProgressChanged 1085
Progressive Abfragen (LINQ) 790
Properties 807, 833
PropertyValueCollection 833
Proxy
  beim HTTP-Request 577, 601
Proxy-Einstellungen
  Internet Explorer 577, 601
Prozesse
  aktivieren 268
  beenden 289
  Laufende ermitteln 269
  personifizieren 845
Prozessor
  Geschwindigkeit 482
  überwachen 514, 516
Prüfen von XML 441
Prüfsumme
  TCP und ICMP 546
Public-Key-Verfahren 873
PushEffect 1002
PushTransform-Methode 1006
PutNextEntry 405

Q
Qualität von Bildern 951
QualityLevel 951, 952
Quantisierungstabellen 945
Quartal
  berechnen 157
  Start und Ende berechnen 157
Queue 116

R
Radiant 55
Radiant siehe Bogenmaß
Rahmenlose Fenster 740
Rahmenlose Forms 635
  bewegen 635
RAM siehe Arbeitsspeicher
Rating-Eigenschaft 922
RawFormat 917
RawValue 525
RC2 857
RC2CryptoServiceProvider 857
Read 383
ReadLine 383
ReadObject-Methode 1069
ReadToEnd 383
Reboot 504
Recherche
  Knowledge Base 34
  MSDN 33
Rechtecke
  Drehpunkt 1011
  mit runden Ecken zeichnen 987
  rotieren 1011
RedirectStandardError 287
RedirectStandardOutput 287
Referenzliste
  Assemblys hinzufügen 403
Reflector 39

Reflektion 299
Regex 77, 94
Region 643
RegisterHotkey 529
Registrierdatenbank siehe Registry
Registry
  Daten auslesen 252
  Daten schreiben 252
RegistryKey 252, 488
Rekursion 355
Relative Pfade
  ermitteln 307
  in absolute wandeln 309
ReleaseCapture 636
Replace 76, 77, 78
ReportProgress 1084
RepositionEditingControlOnValueChange 71
  2
requestedExecutionLevel 221
ResetTransform 1009
Restore
  SQL Server 1120
Return-Taste
  in Tab umwandeln 676, 783
  in TextBox abfangen 675
RichTextBox
  Cursorposition ermitteln 678
Rijndael 857
RijndaelManaged 857
RIPEMD160 881
RNGCryptoServiceProvider 67, 98
Roeder, Lutz 39
Ronald W. Mallen 182
RotateAt 958
RotateTransform 955, 957, 1009
RotateTransform-Klasse 1005
Rotationsmittelpunkt 1007
Rotieren
  von Rechtecken 1011
  von Text 1013
Rotieren von Bildern 955
Round 54
RSA 874
RSACryptoServiceProvider 874
RSA-Verschlüsselung 873
Runden
  kaufmännisch 54
  mathematisch 54

S

Saudiarabischer islamischer Kalender 210
SAX 429
ScaleTransform 955
ScaleTransform-Klasse 1005
Schaltjahr ermitteln 160
Schatten für ein Bild 1001
SchemaClassName 807
Schemas 437
Schlüssel
  Öffentlicher 873
  Privater 873
Schlüsselaustausch
  mit ECDH 878
  per asymmetrischer Verschlüsselung 873
Schnittmenge 115
Schräg zeichnen 1004
Schräges Ausgeben 1004
Schreiben
  Anwendungskonfiguration 230
  Binäre Dateien 392
  Externe Konfiguration 235
  in die Registry 252
  Konfigurationsdaten 230
  Textdateien 385
  von Konfig.-Daten 245
  XML-Dokumente 457, 460, 463, 465
Schreibschutz
  setzen/aufheben 361
Schrittweite 974
Screen 766
Screenshot 938
SE_SHUTDOWN_NAME 505
SE_TCB_NAME 848
section 240
sectionGroup 240
Secure Hash Algorithm 881
SecureString 851
Seicherverbrauch
  reduzieren 226
SelectedPath 722

Send 557
SendKeys 296
SendMessage 636
SendOrPostCallback 1094
Serialisieren 1073
  beliebige (serialisierbare) Objekte 1076
  Binär 1071
  Datenvertrag-Serialisierung 1066, 1068
  Font-Objekte 1076
  Typeface-Objekte 1076
Serialize 1071
Serielle Schnittstellen
  auflisten 496
ServerConnection 1120
ServiceController 272, 276
ServicePack 486
Service-Pack-Version 486
SetComment 405
SetDlgItemText-Funktion 727
SetForegroundWindow 263
SetLevel 405
SetPassword 835
SetProcessWorkingSetSize 228
SetRemapTable 964
SetStyle 640
SetSystemTime 169
Settings 231
SetValue 252, 462
SHA1 881
SHA1Cng 881
SHA1Managed 881
SHA256 881
SHA256Cng 881
SHA256Managed 881
SHA384 881
SHA384Cng 881
SHA384Managed 881
SHA512 881
SHA512Cng 881
SHA512Managed 881
ShadowDepth-Eigenschaft 1002
SharpZipLib 401
ShellLink.tlb 378
ShellShortcut 378
SHFileOperation 321, 371
SHFILEOPSTRUCT 321
Shortcuts siehe Hotkeys
ShowNewFolderButton 722
SHQueryRecycleBin 374, 375
Shutdown 504
Sicherheit
  Asymmetrisches Verschlüsseln 873
  Hashing-Verfahren 879
  Personifizierung 845
  Schlüsselaustausch mit ECDH 878
  Strings sicher verwalten 851
  Symmetrisches Verschlüsseln 855
Sicherheits-Level
  User Account Control 220
Sieb des Eratosthenes XE 1087
Signaturen
  Digitale 873
Simulieren
  der Tab-Taste 676, 783
  von Tasten 296
Size 338
Sjögren, Mattias 378
Skalieren von Bildern 943
SkewTransform-Klasse 1005
Sleep 122
SmtpClient 555
SmtpFailedRecipientException 555
SMTP-Server
  E-Mails vers. 554
SoapException
  auswerten 123
Softness-Eigenschaft 1002
Solarkalender 207
SortedDictionary 116
Sortieren
  Arrays 105
  Auflistungen 105
  ListView 695
  TreeView 700
Sorting 695
Sounddateien
  in WPF absp. 908
  Midi 892, 908
  MP3 892, 908

über MCI absp. 892
Wave 891, 892, 908
Soundkarten
auflisten 502
SoundPlayer 891
SourceInitialized 742
SpecialFolder 303
Spiegeln von Bildern 955
Splash-Fenster (WPF) 761
Splash-Formulare (Windows.Forms) 652
Split 79
Splitten
von Strings 79
Spurverfolgung siehe Tracing
SQL
Datenbanken erz. 1110
in LINQ to SQL direkt ausführen 799
von LINQ to SQL evaluieren 800
SQL Server
Backup und Restore 1120
Datenbanken abfragen 1125
verfügbare ermitteln 1124
SQL Server Query Visualizer 803
SSL-Verbindung 576, 600
StackFrame 1057
StackTrace 1057
Standard User Analyzer 222
Start 276, 283, 284
Startdatum
einer Kalenderwoche 165, 166
einer Woche 153
eines Monats 154
eines Quartals 157
Starten
Anwendung 283, 285
Browser 562
Java-Anwendung 291
StartInfo 562
StartupPath 216, 301
Status
Internetverbindung 541
Steuerelemente
Fokuswechsel durch Cursortasten verhindern 725
Formular passend öffnen 660
Hintergrund mit Textur 748
In einem Thread lesen und schreiben 1079, 1082
in VS integrieren 670
Position relativ zum Bildschirm 659
Transp. Hintergr. 667
Steuerelemente (WPF)
Absolute Position 768
Fenster passend öffnen 770
Position relativ zum Bildschirm 768
Stop 276, 566
Stopwatch 195
StreamGeometry 987
StreamReader 383
StreamWriter 385, 387
Stretch 748
Stride 974
Strings
am Anfang ersetzen 91
Anzahl Wörter 82
auftrennen 79
aus Byte-Array 104
aus Enum 99
beim Zeichnen umbrechen 1022
definiert kürzen 89
entschlüsseln 855, 873
ersetzen 72
gedreht ausgeben 1013
Höhe und Breite 1015
in Byte-Array umw. 101
in Datumswerte konvertieren 213
in Enum-Wert 100
Linker Teilstring 93
mehrf. Leerz. entf. 80
Rechter Teilstring 93
rechtsbündig zeichnen 1018
rotieren 1013
sicher verwalten 851
vergleichen 72
verschlüsseln 855, 873
Wörter extrahieren 85
Wörter normalisieren 81
wortgerecht umbrechen 86
Zahl am Anfang? 95
Zahlen auslesen 94

zentriert zeichnen 1018
Zufalls-Strings 97
StructLayout 321
Subject-Eigenschaft 922
Substring 93
Suchen
Dateien in den Sys.-Pfaden 346
in Arrays 109
in Auflistungen 109
in Newsgroups 30
von Dateien 344
SupportsDaylightSavingTime 205
SymmetricAlgorithm 857, 858
Symmetrisches Verschlüsseln 855
System
Hotkeys auswerten 529
System booten 503
System herunterfahren 503
Systemdatum
lesen 131
setzen 169
SystemDirectory 303, 477
Systemmenü
ändern 639
Schließen-Schalter entf. 637, 639
Systemmonitor 505
Systempfade 477
SystemSounds 71
Systemzeit
setzen 169

T
Tabulator-Taste
bei Return simulieren 676, 783
Tage
im Jahr 148
im Monat 148
TaiwanCalendar 210
Taiwanesischer Kalender 210
TaiwanLunisolarCalendar 210
TargetInvocationException 839
Tastatur
abfragen 723, 786
simulieren 296
Tasten
simulieren 296
Tastenbetätigung
abfragen 723, 786
Tauschen
Knoten in TreeView 698
TCP
Prüfsumme berechnen 546
Teilstrings
am Anfang ersetzen 91
ersetzen 72
Links/Rechts 93
Temporäre Dateien 302
Temp-Ordner ermitteln 303
TextAlignment 1018
TextBox
angezeigte Zeilen auslesen 682
Auto Select 668
Cursorposition ermitteln 678
Eingabe einschränken 670, 776
Return abfangen 675
TextBox (Windows.Forms)
Automatisches Selektieren 668
TextBox (WPF)
Automatisches Selektieren 775
TextChanged-Ereignis
ComboBox (WPF) 781
Textdateien
ändern 389
lesen 383
schreiben 385
Texte anfügen 387
Texte
beim Zeichnen trimmen 1018, 1022, 1024
Texte siehe Strings
TextTrimming.CharacterEllipsis 1022
TextTrimming.None 1022
TextTrimming.WordEllipsis 1022
TextureBrush 964, 969
Texturen 748
transparente 969
ThaiBuddhistCalendar 209
Thailändischer Kalender 209
ThickHorizontalBorderHeight 942
ThickVerticalBorderWidth 942

ThinVerticalBorderWidth 942
Threading
  BackgroundWorker 1082
  Sicher auf Steuerelemente zugreifen 1079, 1082
Threads
  Parameter übergeben 1086
  Rückgabewerte 1086
ThreadStart 1086
Thumbnails 947
Ticks 194
TiffBitmapEncoder 951
TileBrush 748
TileMode 748
TimeZoneInfo 204
Titellose Fenster
  bewegen 741
Title-Eigenschaft 922
ToBase64String 393
ToByte 52
ToInt16 52
ToInt32 52
ToInt64 52
ToLocalTime 203
Tooltip
  in ListBox 693
ToSByte 52
TotalDays 138
TotalFreeSpace 316
TotalHours 138
TotalMilliseconds 138
TotalMinutes 138
TotalSeconds 138
TotalSize 316
ToTitleCase 81
ToUint16 52
ToUint32 52
ToUint64 52
ToUniversalTime 169
Trace-Protokolle 516
Tracing
  RAM- und CPU-Auslastung 516
Transform 474, 957, 958
Transformation
  von Grafikobjekten 1004
Transformationen
  WPF 943, 955
TransformationGroup 955
TransformedBitmap 955
TransformedBitmap-Klasse 943
TransformGroup-Klasse 1005
Transformieren
  Bilder in WPF 979
  mit LINQ to XML 471
  Pixel 966, 981
  XML-Dateien 473
  Zeichenmatrix 957
TransformToDevice 765
Translate 958
TranslatePoint 768
TranslateTransform 957, 1008
TranslateTransform-Klasse 1005
Transp. Hintergrund 642
Transparente
  Bilder 996
  Steuerelemente 667
TreeView
  Knoten sortieren 700
  Knoten tauschen 698
Trimming 1018, 1022
Trimmung
  von gezeichneten Texten 1018, 1022, 1024
TripleDES 857
TripleDESCryptoServiceProvider 857
TryParse-Methode 62
Typ
  eines Laufwerks ermittteln 313
Typeface 1076
Typisierte Auflistungen 115
Typisierte Dictionaries 115
Typisierte Hashtables 115

U

UAC siehe User Account Control
Überprüfen
  Datei-Existenz 344
  Gültigkeit Datum 65
  Gültigkeit v. Zahlen 61
  Ordner-Existenz 317

von Pfadangaben 310
Zeit bei der Eingabe 201
Übertragungsbestätigung
bei SMTP-E-Mail 558
Überwachen
Arbeitsspeicher 511, 514, 516
CPU 514, 516
Dateisystem 380
Uhrzeit
Aktuelle 131
UmAlQuraCalendar 210
Umbenennen
von Dateien 351
von Ordnern 336
Umbrechen
Strings beim Zeichnen 1022
von Strings 86
Umleiten
von Methodenaufrufen 1080, 1081
Umm-Al-Qura-Kalender 210
Unable to relay for 555
Unbewegbare Formulare 637
Undo 845
Ungleichheits-Verknüpfungen 793
UnhandledException 128
Unicode-Zeichensatz 1136
Upgrade-Methode (WPF) 233
Upload
Dateien (Webserver) 601, 614
mit BITS 601
UploadData 614
UploadFileAsync 615
UploadFileCompleted 618
UploadProgressChanged 618
UploadString 615
UploadStringAsync 615
User Account Control 219
Anhebung des Sicherheits-Level 220
Drag&Drop 220
Einschränkungen 220
User siehe Benutzer
UserName 844
UseShellExecute 284, 287
using 227, 903

V
ValidationEventHandler 452
ValidationType 441
Validieren
Datum bei der Eingabe 197
von XML 441
Zeit bei der Eingabe 201
ValidKeySize 863
Value 434
ValueType 714
Verbindung
geschützte 576, 600
zum Internet überprüfen 544
Verfügbare Kulturen
auslesen 57
Verglasen
eines Fensters 753
Vergleichen
Datumswerte 137
von Dateien 353
von Ordnern 355
von Strings 72
Verknüpfungen
Ungleichheits- 793
Verknüpfungen anlegen 377
Verlaufendes Füllen
eines Fensters 747
Verschieben
von Dateien 351
von Ordnern 337
Verschlüsseln
Asymmetrisch 873
ECDH 878
Hashcodes 879
RSA 873
Symmetrisch 855
Versenden
von E-Mails 554, 560
Version
des Service Pack 486
einer Datei auslesen 363
Windows-Vers. 480
Verzeichnis siehe Ordner
Verzeichnisse
FTP 621

Videodateien
AVI 892, 908
MPEG 892, 908
Videos
in WPF absp. 908
über MCI absp. 892
ViewPort 749
ViewPortUnits 749
Vista-Glas-Effekt 647, 753
Visual Studio
Steuerelem. integr. 670

W
W3Schools 436
WaitForExit 285, 290
WaitForPendingFinalizers-Methode (GC-Klasse) 228
WaitForStatus 276
Wave-Dateien
abspielen 891
in WPF absp. 908
über MCI absp. 892
WebBrowser 563
WebClient 569
WebRequest 578
WebResponse 578
Webseiten
in Anwendung 563
in Browser 562
Webserver
Download 569, 578
Upload 614
Websites
zu C# und .NET 27
Win32_OperatingSystem 483, 503
Win32_ParallelPort 491
Win32_Processor 482
Win32_Product 488
Win32_SerialPort 496
Win32_SoundDevice 502
WindowCaptionHeight 942
Windows
booten 503
Hauptversion 480
herunterfahren 503
Hotkeys auswerten 529
Systempfade 477
Version auslesen 480
Windows.Forms-Anwendungen
Ausnahmen global abfangen 223
Globale Ausnahmebehandlung 225
WindowsIdentity 845
WindowsImpersonationContext 845
Windows-Nachrichten
in WPF verarbeiten 741
Windows-Nachrichten XE 728
Windows-Ordner ermitteln 303
WindowStyle 283
Winkel
in Bogenmaß 55
WinNT-ADSI-Provider 805, 817, 823
WM_GETMINMAXINFO 743
WM_HOTKEY 529
WMI
CPU-Geschwindigkeit 482
Installierte Progr. listen 488
Parallele Schnittst. listen 491
RAM-Infos 483
Serielle Schnittst. listen 496
Soundkarten listen 502
WmpBitmapEncoder 951
WndProc 635
Woche
Kalenderwoche berechnen 161
Start einer Kalenderwoche 165
Start und Ende berechnen 153
Wochentag
ermitteln 149
Name ermitteln 150
Word-Dokumente
erzeugen 1037
öffnen 1033
Workbooks 1042
Working Set 228
Worksheets 1042
Wörter
Anzahl in String 82
aus einem String 85
erstes Zeichen groß 81

WPF
Bilder transformieren 979
Multimediadateien absp. 908
Transformationen 943, 955, 1004
WPF-Anwendungen
Ausnahmen global abfangen 223
Write 386
WriteAttributeString 458
WriteBase64 458, 469
WriteChars 458
WriteElementString 458
WriteEndDocument 458
WriteEndElement 458
WriteObject-Methode 1068
WriteStartDocument 458
WriteStartElement 458
WriteXml 465

X
XContainer 434
XDocument 424, 433, 453, 460
XElement 424, 433, 434, 453, 460
XML validieren 441
XML
Namensräume 427
XPath-Navigation 453, 455
XML Paper Specification 1033
XmlDictionaryReader-Klasse 1070
XmlDictionaryWriter-Klasse 1070
XmlDocument
XML validieren 441
XML-Dokumente
ändern 467
Binäre Daten 390
binäre Daten lesen 470
Gültigkeit überprüfen 441
mit LINQ to XML erzeugen 463
mit LINQ to XML transformieren 471
über das X-DOM erzeugen 462
über das X-DOM lesen 433
über einen XmlReader lesen 429
über LINQ lesen 423
validierend lesen 436
XmlNamespaceManager 455, 456
xmlns 427
XmlReader 429, 437, 441
XmlReaderSettings 437, 445
XmlResolver 444
XML-Schemas 436
XmlSerializer 1073
XmlUrlResolver 444
XmlValidatingReader 451
XmlWriter 457
XmlWriterSettings 458
XName 428
XNamespace 428
XNode 434
XPath 453
XPathSelectElement 453
XPathSelectElements 453
XPath-Suche
mit Namensraum 455
ohne Namensraum 453
XPS-Dokument 1033
XSL 473
XslCompiledTransform 474

Z
Zahlen
am Stringanfang? 95
aus String auslesen 94
dual darstellen 46
formatieren 43
größter und kleinster Wert 55
Gültigkeit 61
hexadezimal darst. 45
High- u. Low-Word 69, 70
in TextBox 670, 776
kaufmännisch runden 54
länderspezifisch 60
mathematisch runden 54
negative anders formatieren 45
Notationen 52
von Dual 51
von Hexadezimal 46
Zufallszahlen 66
Zweierkomplement 46
Zeichenketten siehe Strings
Zeichnen
Graustufen 979

Negativ 971
Pfeile 991
Probleme vermeiden 983
Rechtecke mit runden Ecken 987
schräg 1004
Strings, rechtsbündig 1018
Strings, zentriert 1018
Transformieren 966
Transparent 996
Zeilen
angezeigte aus einer TextBox 682
Zeilenabstand
beim Zeichnen von Texten 1024
Zeit
Eingabe validieren 201
genau messen 194
Systemzeit setzen 169
Zeitangabe
in DateTime konvertieren 147
Zeitzonen 204
Offset berechnen 206
Zellen
eines DataGridView 702
ZIP 399, 401, 407
ZIP-Algorithmus 395
ZIP-Archive erzeugen 409
ZipEntry 405
ZipOutputStream 404
ZipPackage 395, 399
Zufalls-String 97
Zufallszahlen 66
Zugriffsdaten
einer Datei 362
Zulu Time 136
Zweierkomplement 47
Zwischenablage
abfangen 774
Bitmaps lesen 936, 937



informit.de, Partner von
Addison-Wesley, bietet aktuelles
Fachwissen rund um die Uhr.

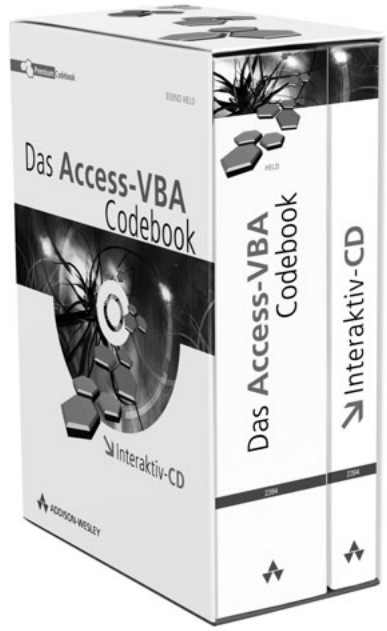
www.informit.de

In Zusammenarbeit mit den Top-Autoren von Addison-Wesley, absoluten Spezialisten ihres Fachgebiets, bieten wir Ihnen ständig hochinteressante, brandaktuelle deutsch- und englischsprachige Bücher, Softwareprodukte, Video-Trainings sowie eBooks.

wenn Sie mehr wissen wollen …

www.informit.de

THE SIGN OF EXCELLENCE



Dieses Premium-Codebook liefert zahlreiche, sofort verwertbare Programmbeispiele zu allen Gebieten der Access-VBA-Programmierung. Die Einteilung in Kategorien ermöglicht es, alle enthaltenen Rezepte schnell aufzufinden. Die beschriebenen Funktionen können leicht in jede Access-Anwendung integriert werden und sparen Ihnen Entwicklungszeit für immer wiederkehrende Aufgaben aus dem Alltag des Programmierers.

Bernd Held
ISBN 978-3-8273-2394-1
99.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence] ADDISON-WESLEY

THE SIGN OF EXCELLENCE



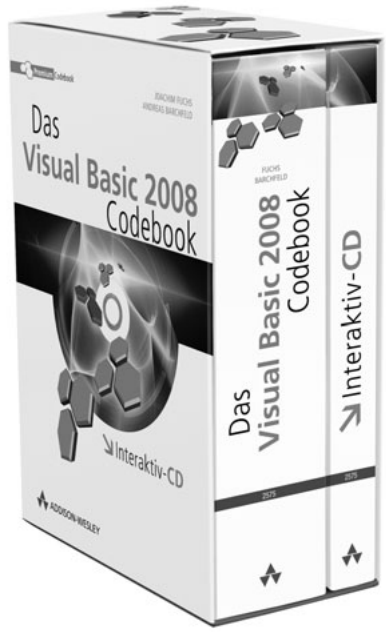
Das Excel-VBA-Codebook liefert Tipps, Tricks und Lösungen zu allen Problemen der täglichen Programmierarbeit mit den Excel-Versionen 2000 bis 2007. Alle diese „Rezepte“ sind in Kategorien sortiert und somit sehr leicht auffindbar. Die CD zum Buch enthält nochmals alle vorgestellten Codes, sodass diese sehr schnell in eigene Projekte eingefügt werden können.

Melanie Breden; Michael Schwimmer
ISBN 978-3-8273-2471-9
99.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence] ADDISON-WESLEY

THE SIGN OF EXCELLENCE



Das Buch liefert dem Leser fertige Lösungen zu praxisbezogenen, täglich auftretenden Programmierproblemen. Alle diese „Rezepte“ sind in Kategorien sortiert und somit sehr leicht auffindbar. Die CD zum Buch enthält nochmals alle vorgestellten Codes, sodass diese sehr schnell in eigene Projekte eingefügt werden können.

Joachim Fuchs; Andreas Barchfeld
ISBN 978-3-8273-2575-4
99.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
ADDISON-WESLEY